



Liberty ID-WSF People Service Specification

Version: 1.0

Editors:

Yuzo Koga, Nippon Telegraph and Telephone Corporation
Paul Madsen, Nippon Telegraph and Telephone Corporation

Contributors:

Robert Aarts, Hewlett-Packard
Conor Cahill, America Online, Inc.
Carolina Canales-Valenzuela, Ericsson
Scott Cantor, Internet2 / The Ohio State University
Peter Davis, NeuStar, Inc.
Jeff Hodges, NeuStar, Inc.
Greg Whitehead, Hewlett-Packard

Abstract:

The Liberty Identity Web Services Framework (ID-WSF) supports the discovery and invocation of identity services - web service interfaces exposed on behalf of a user.

There exist many circumstances where a user may wish to access the identity resources (either browser-based or service-based) of another user. Some examples include: a parent wishing to discover the current location of their child, someone wishing to share photographs stored at some service with their friends, or allowing one game-player to determine whether another player is available.

In such cases, it is necessary for one user (or a provider acting on their behalf) to be able to obtain an appropriate identifier for another user from that user's Identity Provider, and to convey that identifier to this second user's identity services.

Additionally, users will often desire to grant access rights to both browser-based resources as well as their identity services to friends and colleagues - this implies that the privileges can be assigned to a relevant identifier for that friend as supplied by an appropriate identity provider.

This document describes an architecture for enabling secure, privacy-respecting *cross-principal* online interactions between users and the identity resources (both browser-based and programmatic services) of others, and normatively defines the Liberty ID-WSF People Service to support such interactions.

Ultimately, such cross-principal interactions will depend of a variety of mechanisms and components of the full ID-WSF architecture beyond the People Service alone.

Filename: liberty-idwsf-people-service-v1.0.pdf

1

Notice

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

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

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

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

34 Contents

35	1. Introduction	6
36	1.1. Overview	6
37	1.2. Notation	6
38	1.3. Terminology	7
39	1.4. Namespaces	7
40	2. Data Model	8
41	2.1. <Object> Element	8
42	2.1.1. NodeType Attribute	8
43	2.1.2. CreatedDateTime Attribute	8
44	2.1.3. ModifiedDateTime Attribute	8
45	2.1.4. <ObjectID> Element	9
46	2.1.5. <DisplayName> Element	9
47	2.1.6. <Tag> Element	9
48	2.1.7. <ObjectRef> Element	9
49	2.2. <Token> Element	10
50	3. People Service	11
51	3.1. Overview	11
52	3.2. Service Type	11
53	3.3. Action URIs	11
54	3.4. Request and Response Abstract Types	12
55	3.4.1. Complex Type RequestAbstractType	12
56	3.4.2. Complex Type ResponseAbstractType	12
57	3.5. Status	12
58	3.6. Identity Token Policy	14
59	3.7. Success & Failure	14
60	3.8. Subscription and Notification	14
61	3.8.1. <Subscription> Element	14
62	3.8.2. Notify and NotifyResponse Messages	15
63	3.8.3. <Notification> Element	16
64	3.9. Adding an Entity	16
65	3.9.1. wsa:Action Values	17
66	3.9.2. AddEntityRequest Message	17
67	3.9.3. AddEntityResponse Message	18
68	3.9.4. Processing Rules	19
69	3.10. Adding a Known Entity	20
70	3.10.1. wsa:Action Values	21
71	3.10.2. AddKnownEntityRequest Message	21
72	3.10.3. AddKnownEntityResponse Message	22
73	3.10.4. Processing Rules	23
74	3.11. Removing an Entity	24
75	3.11.1. wsa:Action Values	24
76	3.11.2. RemoveEntityRequest Message	24
77	3.11.3. RemoveEntityResponse Message	25
78	3.11.4. Processing Rules	26
79	3.12. Adding a Collection	26
80	3.12.1. wsa:Action Values	26
81	3.12.2. AddCollectionRequest Message	26
82	3.12.3. AddCollectionResponse Message	27
83	3.12.4. Processing Rules	28
84	3.13. Removing a Collection	28
85	3.13.1. wsa:Action Values	28
86	3.13.2. RemoveCollectionRequest Message	28

87	3.13.3. RemoveCollectionResponse Message	29
88	3.13.4. Processing Rules	29
89	3.14. Adding to a Collection	29
90	3.14.1. wsa:Action Values	30
91	3.14.2. AddToCollectionRequest Message	30
92	3.14.3. AddToCollectionResponse Message	31
93	3.14.4. Processing Rules	31
94	3.15. Removing from a Collection	31
95	3.15.1. wsa:Action Values	31
96	3.15.2. RemoveFromCollectionRequest Message	31
97	3.15.3. RemoveFromCollectionResponse Message	32
98	3.15.4. Processing Rules	33
99	3.16. Listing Members	33
100	3.16.1. wsa:Action Values	33
101	3.16.2. ListMembersRequest Message	33
102	3.16.3. ListMembersResponse Message	35
103	3.16.4. Examples	36
104	3.16.5. Processing Rules	38
105	3.17. Retrieving Info	39
106	3.17.1. wsa:Action Values	39
107	3.17.2. GetObjectInfoRequest Message	39
108	3.17.3. GetObjectInfoResponse Message	40
109	3.17.4. Processing Rules	40
110	3.18. Updating Info	41
111	3.18.1. wsa:Action Values	41
112	3.18.2. SetObjectInfoRequest Message	41
113	3.18.3. SetObjectInfoResponse Message	42
114	3.18.4. Processing Rules	42
115	3.19. Querying Objects	43
116	3.19.1. wsa:Action Values	43
117	3.19.2. QueryObjectsRequest Message	43
118	3.19.3. QueryObjectsResponse Message	44
119	3.19.4. Processing Rules	45
120	3.20. Testing Membership	46
121	3.20.1. wsa:Action Values	46
122	3.20.2. TestMembershipRequest Message	46
123	3.20.3. TestMembershipResponse Message	47
124	3.20.4. Processing Rules	48
125	3.21. Resolving Objects	48
126	3.21.1. wsa:Action Values	48
127	3.21.2. ResolveIdentifierRequest Message	48
128	3.21.3. ResolveIdentifierResponse Message	49
129	3.21.4. Processing Rules	50
130	4. Interaction with Users	52
131	4.1. Model (Informative)	52
132	4.2. Additional Federations for Sharing of Identity Services	52
133	4.3. Consent Model	53
134	4.4. Elements Supporting Invitation	53
135	4.4.1. PStoSPRedirectURL element	53
136	4.4.2. <SpttoPSRedirectURL> element	53
137	4.4.3. <QueryString> element	54
138	5. Sequence Examples	58
139	5.1. Policy definition	58
140	5.2. AccessControl	58

141	5.3. Group Operation	60
142	6. Security Considerations	64
143	7. XML Schema for ID-WSF People Service	65
144	8. Abstract WSDL	74
145	References	80

146 **1. Introduction**

147 **1.1. Overview**

148 A user's People Service (PS) is an interface into those other users with which the owning user wishes to (or has
149 already) interact with in some online fashion - these other users possibly categorized into arbitrary `groups`. The PS
150 provides a flexible, privacy respecting framework by which a user can manage/track the people they know and how
151 these other users are related.

152 The first generation of online transactions/interactions were single-user, eg. online banking, travel booking, shopping
153 etc. More and more however, our online interactions involve other users than just ourselves. Whether it is
154 communication, commerce, sharing, self-expression, or collaboration being enabled - all these interactions build on a
155 social layer that connects individuals to others. Unfortunately, the current situation is that each of these applications
156 generally builds its view of a given individual's complete social network. This can result in duplication and undesirable
157 management burden on those individuals, forced to maintain these multiple views.

158 Many interesting interactions will involve those individuals who are both explicit and direct. For instance, a user
159 may wish to share their online photos with their family, or they may need to determine the network presence of their
160 colleagues.

161 Enabling such direct interactions between users and their circle of friends is straightforward when both maintain an
162 account at the same provider. On many online photo sites for instance, users share their photos with others but only
163 once they have established an account at the same provider. If the first user already knows the account name of the
164 other, all that need happen is for that name to be supplied. If they don't know it, they might search existing accounts
165 or, if necessary, have an invite sent to their friend encouraging them to create an account.

166 There are two significant implications of this model:

167 1. Both users must maintain or establish accounts at the same provider. Typically, the result of this requirement
168 is that the friend being invited to interact (e.g. View vacation photos, etc) is forced to create an account (with
169 associated logins and passwords to remember) at a provider where they might not otherwise choose to do so.

170 2. If some connection between two friends is established in the context of the photo site, it can't be leveraged in
171 some other context (e.g. Calendar sharing) unless that provider happens to host both services.

172 Enabling such cross-user interactions such that the above two implications are addressed is the goal of the Liberty
173 Alliance's People Service. The People Service provides a flexible, privacy respecting framework by which one user
174 can manage/track the people they know - typically but not exclusively in order to assign them certain privileges for
175 accessing certain resources owned by the first user. Providers query/manipulate this information through standardized
176 interfaces.

177 Additionally, to satisfy the requirement for informing a user of another's intent to add them to their PS resource, an
178 invitation model by which user's can be informed of such and establish the necessary federations between providers is
179 defined.

180 This document is the Liberty Identity Web Services Framework (ID-WSF) People Service Specification that norma-
181 tively specifies the People Service protocols.

182 **1.2. Notation**

183 This specification uses schema documents conforming to W3C XML Schema (see [[Schema1-2](#)]) and normative text
184 to describe the syntax and semantics of XML-encoded protocol messages. Note: Phrases and numbers in brackets []
185 refer to other documents; details of these references may be found at the end of this document.

186 The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT,"
187 "RECOMMENDED," "MAY," "MAY NOT," and "OPTIONAL" in this specification are to be interpreted as described

188 in [RFC2119]: "they MUST only be used where it is actually required for interoperability or to limit behavior which
189 has potential for causing harm (e.g., limiting retransmissions)."

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

193 This specification uses the following typographical conventions in text: <Element>, <ns:ForeignElement>,
194 attribute, Datatype, and OtherCode.

195 Definitions for Liberty-specific terms may be found in [LibertyGlossary].

196 1.3. Terminology

197 The Liberty terms *Service Provider* and *Web Service Consumer*, and their respective abbreviations, SP and
198 WSC, refer to different roles that may be assumed by the same website. Generally, an SP is some website that
199 provides online services to users through HTTP interactions. In interactions with other providers not mediated by a
200 user's browser, websites assume the role of a WSC in order to send SOAP-based requests. For clarity, this specification
201 uses the SP abbreviation to refer to both these rules, distinguishing where appropriate.

202 1.4. Namespaces

203 The following namespaces are used in the schema definitions:

- 204 • The prefix `ps:` stands for the Liberty ID-WSF People Service schema namespace (`urn:liberty:ps:2006-08`).
205 This namespace is the default for instance fragments, type names, and element names in this document.
- 206 • The prefix `xs:` stands for the W3C XML schema namespace (`http://www.w3.org/2001/XMLSchema`)
207 [Schema1-2].
- 208 • The prefix `xml:` stands for the W3C XML namespace (`http://www.w3.org/XML/1998/namespace`) [XML].
- 209 • The prefix `saml:` stands for the OASIS SSTC SAML2.0 Assertion namespace (`urn:oasis:names:tc:SAML:2.0:assertion`)
210 [SAMLCore2].
- 211 • The prefix `samlp:` stands for the OASIS SSTC SAML2.0 Protocol namespace (`urn:oasis:names:tc:SAML:2.0:protocol`)
212 [SAMLCore2].
- 213 • The prefix `ims:` stands for the Liberty ID-WSF Authentication Service Identity Mapping Service namespace
214 (`urn:liberty:ims:2006-08`) [LibertyAuthn].
- 215 • The prefix `sec:` stands for the Liberty ID-WSF Security Mechanisms Core namespace (`urn:liberty:sec:2005-11`)
216 [LibertySecMech].
- 217 • The prefix `subs:` stands for the Liberty ID-WSF Subscriptions & Notifications namespace
218 (`urn:liberty:ssos:2006-08`) [LibertySUBS].

219 2. Data Model

220 A given user's PS holds information about those other users with which the owning user may have established some
221 online relationship. The owning user may also choose to organize these other users into groups (e.g. their teammates
222 on a hockey team). The PS data model defines how these users and groups are represented.

223 2.1. <Object> Element

224 Both individual users and the groups to which they may belong are represented as <Object> elements - whether an
225 <Object> refers to a group or a user (or perhaps some other individual entity) is distinguished by a `NodeType` attribute
226 with values of `urn:liberty:ps:collection` or `urn:liberty:ps:entity` respectively (see [Section 2.1.1](#) for exact definition).

227 The <Object> element has <DisplayName> elements to carry a human-readable name for the <Object> (see
228 [Section 2.1.5](#)).

229 The <ObjectID> element uniquely labels each <Object> (see [Section 2.1.4](#)).

230 The optional `CreatedDateTime` and `ModifiedDateTime` attributes express the time at which an Object was
231 created and last modified respectively (see [Section 2.1.2](#)).

232 To account for nested Objects, an <Object> element can have multiple <Object> and/or <ObjectRef> elements
233 to refer to other Objects.

234 The schema model for the <Object> element is shown below.

```
235 <xs:element name="Object" type="ObjectType" />
236 <xs:complexType name="ObjectType">
237   <xs:sequence>
238     <xs:element ref="ObjectID" minOccurs="0" />
239     <xs:element name="DisplayName" type="LocalizedDisplayNameType"
240       minOccurs="1" maxOccurs="unbounded" />
241     <xs:element name="Tag" type="TagType" minOccurs="0" />
242     <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded" />
243     <xs:element name="ObjectRef" type="ObjectIDType" minOccurs="0" maxOccurs="unbounded" />
244   </xs:sequence>
245   <xs:attribute name="NodeType" type="xs:anyURI" use="required" />
246   <xs:attribute name="CreatedDateTime" type="xs:dateTime" use="optional" />
247   <xs:attribute name="ModifiedDateTime" type="xs:dateTime" use="optional" />
248 </xs:complexType>
```

251 2.1.1. NodeType Attribute

252 The `NodeType` attribute is defined such that the WSC can distinguish if an <Object> refers to a group or a user (or
253 some other individual entity). For the values of the `NodeType` attribute, the following two URI's are defined:

254 *urn:liberty:ps:collection* If an <Object> has this URI for the value of the `NodeType` attribute, it repre-
255 sents a collection that has zero or more <Object> as child elements. The child
256 <Object> elements may have a `NodeType` of either `urn:liberty:ps:collection` or
257 `urn:liberty:ps:entity`.

258 *urn:liberty:ps:entity* If an <Object> has this URI for the value of the `NodeType` attribute, it represents a single
259 entity (e.g. a user). An <Object> with a `NodeType` of `urn:liberty:ps:entity` MUST
260 NOT itself contain any child <Object> or <ObjectRef> elements.

261 **2.1.2. CreatedDateTime Attribute**

262 The CreatedDateTime attribute may be used by a PS provider to set the time when an <Object> is instantiated.

263 **2.1.3. ModifiedDateTime Attribute**

264 The ModifiedDateTime attribute may be used by a PS provider to set the time when the data or attributes that an
265 <Object> has are changed.

266 **2.1.4. <ObjectID> Element**

267 The <ObjectID> element is defined so that the PS provider can scope each member's identifier locally and uniquely.

```
268  
269 <!-- Declaration of ObjectID element -->  
270 <xs:element name="ObjectID" type="ObjectIDType" />  
271 <!-- Definition of ObjectIDType -->  
272 <xs:complexType name="ObjectIDType">  
273 <xs:simpleContent>  
274 <xs:restriction base="xs:anyURI" />  
275 </xs:simpleContent>  
276 </xs:complexType>  
277
```

278 Where privacy is a concern, PS providers MUST ensure that ObjectID's do not create a privacy concern by allowing
279 different WSCs to make inappropriate correlations about the users for which the Object identifiers stand. Unique
280 identifiers for different WSCs (e.g. pairwise identifiers) and encrypted identifiers are potential mechanisms for
281 addressing this concern.

282 **2.1.5. <DisplayName> Element**

283 The <DisplayName> element provides a human-readable friendly name for Objects. The value of this element
284 SHOULD NOT be used to uniquely identify Objects; rather the ObjectID element SHOULD be used. (see
285 [Section 2.1.4](#)).

```
286  
287 <xs:complexType name="LocalizedDisplayNameType">  
288 <xs:simpleContent>  
289 <xs:extension base="xs:string">  
290 <xs:attribute name="Locale" type="xs:language" use="required" />  
291 <xs:attribute name="IsDefault" type="boolean" use="optional" />  
292 </xs:extension>  
293 </xs:simpleContent>  
294 </xs:complexType>  
295
```

296 **2.1.6. <Tag> Element**

297 The <Tag> element allows users to add their own metadata to <Object> elements. For instance, a user might add a
298 <Tag> element with a value of 'sports' for the Ref attribute to a group Object called 'Team' to denote the theme of
299 that group (perhaps to distinguish it from another group with the same name for some work project).

```
300  
301 <xs:complexType name="TagType">  
302 <xs:attribute name="Ref" type="xs:anyURI" use="required" />  
303 </xs:complexType>  
304
```

305 The value of the Ref attribute SHOULD be a tag space (a place that collates or defines tags), where the last component
306 of the URL is the tag. For instance, *http://technorati.com/tag/music* is a URL for the tag "music."

307 **2.1.7. <ObjectRef> Element**

308 The <ObjectRef> element is used as a pointer to an <Object> through that <Object>'s <ObjectID> element.

309 The <ObjectRef> element allows an <Object> element to be included in another by reference rather than directly.
310 For instance, the fact that a given user belongs to different groups can be represented by both those groups' <Object>
311 element containing an <ObjectRef> element that refers to that user's <Object> element.

312 **2.2. <Token> Element**

313 The <sec:Token> element acts as a container for identity tokens, see [[LibertySecMech](#)]

314 The <sec:Token> element is used by the PS provider to return requested identity tokens to the WSC, either in a
315 <ResolveIdentifierResponse> message or in a <Notify> message to a previous <Subscription> element.

316 **3. People Service**

317 **3.1. Overview**

318 A People Service is an ID-WSF identity web service by which service consumers can query the list of entities (e.g.
319 friends, co-workers, family, devices etc) with which a particular individual chooses to track an online relationship.
320 These listed individual may be organized into groups. Service consumers use the People Service to add members
321 and/or groups, update information for particular members or groups, test group membership of a particular user, and
322 obtain identity tokens for desired members.

323 **3.2. Service Type**

324 A People Service is identified by the service type URN:

325 *urn:liberty:ps:2006-08*

326 **3.3. Action URIs**

327 WS-Addressing defines the <Action> header by which the semantics of an input, output, or fault message can be
328 expressed.

329 This specification defines the following action identifiers:

- 330 • *urn:liberty:ps:2006-08:AddEntityRequest*
- 331 • *urn:liberty:ps:2006-08:AddEntityResponse*
- 332 • *urn:liberty:ps:2006-08:AddKnownEntityRequest*
- 333 • *urn:liberty:ps:2006-08:AddKnownEntityResponse*
- 334 • *urn:liberty:ps:2006-08:RemoveEntityRequest*
- 335 • *urn:liberty:ps:2006-08:RemoveEntityResponse*
- 336 • *urn:liberty:ps:2006-08:AddCollectionRequest*
- 337 • *urn:liberty:ps:2006-08:AddCollectionResponse*
- 338 • *urn:liberty:ps:2006-08:RemoveCollectionRequest*
- 339 • *urn:liberty:ps:2006-08:RemoveCollectionResponse*
- 340 • *urn:liberty:ps:2006-08:AddToCollectionRequest*
- 341 • *urn:liberty:ps:2006-08:AddToCollectionResponse*
- 342 • *urn:liberty:ps:2006-08:RemoveFromCollectionRequest*
- 343 • *urn:liberty:ps:2006-08:RemoveFromCollectionResponse*
- 344 • *urn:liberty:ps:2006-08:ListMembersRequest*
- 345 • *urn:liberty:ps:2006-08:ListMembersResponse*
- 346 • *urn:liberty:ps:2006-08:GetObjectInfoRequest*

- 347 • *urn:liberty:ps:2006-08:GetObjectInfoResponse*
- 348 • *urn:liberty:ps:2006-08:SetObjectInfoRequest*
- 349 • *urn:liberty:ps:2006-08:SetObjectInfoResponse*
- 350 • *urn:liberty:ps:2006-08:QueryObjectsRequest*
- 351 • *urn:liberty:ps:2006-08:QueryObjectsResponse*
- 352 • *urn:liberty:ps:2006-08:TestMembershipRequest*
- 353 • *urn:liberty:ps:2006-08:TestMembershipResponse*
- 354 • *urn:liberty:ps:2006-08:ResolveIdentifierRequest*
- 355 • *urn:liberty:ps:2006-08:ResolveIdentifierResponse*
- 356 • *urn:liberty:ps:2006-08:Notify*
- 357 • *urn:liberty:ps:2006-08:NotifyResponse*

358 **3.4. Request and Response Abstract Types**

359 **3.4.1. Complex Type RequestAbstractType**

360 All PS request messages are of types that are derived from the abstract **RequestAbstractType** complex type.

361 **anyAttribute** [**Optional**] An attribute from a namespace other than that of this specification.

362 The following schema fragment defines the XML **RequestAbstractType** complex type:

```
363 <!-- Definition of RequestAbstractType -->
364 <xs:complexType name="RequestAbstractType" abstract="true">
365   <xs:anyAttribute namespace="##other" processContents="lax"/>
366 </xs:complexType>
367
368
```

369 **3.4.2. Complex Type ResponseAbstractType**

370 All PS response messages are of types that are derived from the abstract **ResponseAbstractType** complex type.

371 This type defines common attributes and elements that are associated with all PS responses:

372 **<lu:Status>** [**Required**] The **<lu:Status>** element is used to convey status codes and related information. The
373 schema fragment is defined in the Liberty ID-WSF Utility schema. The local definition of
374 status codes are described in [Section 3.5](#).

375 **anyAttribute** [**Optional**] An attribute from a namespace other than that of this specification.

376 The following schema fragment defines the XML **ResponseAbstractType** complex type:

```
377
378 <!-- Definition of ResponseAbstractType -->
379 <xs:complexType name="ResponseAbstractType" abstract="true">
380   <xs:sequence>
381     <xs:element ref="lu:Status"/>
382   </xs:sequence>
383   <xs:anyAttribute namespace="##other" processContents="lax"/>
384 </xs:complexType>
385
```

386 **3.5. Status**

387 All the response messages extended from `ResponseAbstractType` contain a `<lu:Status>` element (see
388 [Section 3.4.2](#)) to indicate whether or not the processing of the request message has succeeded. The `<lu:Status>`
389 element is included from the Liberty ID-WSF Utility Schema. A `<lu:Status>` element MAY contain other
390 `<lu:Status>` elements providing more detailed information. A `<lu:Status>` element has a `code` attribute,
391 which contains the return status as a string. The local definition of these codes is specified in this document. This
392 specification defines the following status codes to be used as values for the `code` attribute:

- 393 • `CannotFindIDP`
- 394 • `CannotFindObject`
- 395 • `CannotResolveToken`
- 396 • `Failed`
- 397 • `InvalidNodeType`
- 398 • `InvalidObjectID`
- 399 • `ObjectIsCollection`
- 400 • `ObjectIsEntity`
- 401 • `OK`
- 402 • `NoResults`
- 403 • `PartialSuccess`
- 404 • `PolicyDoesNotAllow`
- 405 • `ResolveIdentifierNotSupported`
- 406 • `Timeout`
- 407 • `UnexpectedError`
- 408 • `UnrecognizedFilter`
- 409 • `UnrecognizedNamespace`
- 410 • `UnspecifiedError`

411 The `<lu:Status>` element may contain other `<lu:Status>` elements supplying more detailed return status infor-
412 mation. The code attribute of the top level `<lu:Status>` element MUST contain either *OK*, *PartialSuccess*, or *Failed*.
413 The remainder of the values above are used to indicate more detailed return status inside second level `<lu:Status>`
414 element(s).

415 *OK* The value *OK* means that the processing of the request message has succeeded. A second
416 level status code MAY be used to indicate some special cases, but the processing of the
417 request message has succeeded.

418 *PartialSuccess* The value *PartialSuccess* means that the processing of the request message has partially
419 succeeded. A second level status code MAY be used to indicate which processes failed to be
420 processed.

421 *Failed* The value *Failed* means that the processing of the request message has failed. A second level
422 status code SHOULD be used to indicate the reason for the failure.

423 **3.6. Identity Token Policy**

424 For those messages that may result in an identity token being returned (either directly or not) to the WSC, that WSC
425 may wish to indicate its requirements of that identity token. For instance, the WSC may wish that the returned identity
426 token should carry a long-lived federated identifier for the user in question. Alternatively, should its immediate
427 requirements not justify the establishment of such a federated identifier (and the potential associated management
428 burden) it may desire only a short-lived and transient identifier.

429 The `<sec:TokenPolicy>` element serves as a container for such WSC policy requirements. The
430 `<sec:TokenPolicy>` element is defined in [[LibertySecMech](#)]

431 If no `<sec:TokenPolicy>` element is present, or if there is no `<NameIDPolicy>` element within a
432 `<sec:TokenPolicy>` element, the default identity token policy is that the WSC desires a SAML assertion
433 with a name identifier with a format of *urn:oasis:names:tc:SAML:2.0:nameid-format:persistent*.

434 If the WSC desires an alternative identity token, it MUST specify this accordingly.

435 **3.7. Success & Failure**

436 Except for the `ResolveIdentifierRequest` message, for those protocol messages that support multiple operations to be
437 requested in a single message (e.g. removing multiple users from a targeted group in one step), all operations succeed
438 or fail together.

439 **3.8. Subscription and Notification**

440 When present in a PS request message, a `<Subscription>` element indicates that the WSC wishes to be notified if and
441 when the data associated with the relevant `Object` (either being created or targeted through a `<TargetObjectID>`)
442 changes.

443 For each request message for which a `<Subscription>` element is allowed, this specification defines the `Object` for
444 which changes are being subscribed to and the data that the PS provider will return in any `<Notify>` message.

445 The subscription & notification model used within this specification can be considered a constrained version of the
446 more flexible model defined in [[LibertySUBS](#)].

447 **3.8.1. `<Subscription>` Element**

448 It is by including a `<Subscription>` element in a request message that a WSC subscribes to be notified if and when
449 the object created or targeted by that request message changes. The contents of the `<Subscription>` element gives
450 the WSC some control over the parameters of the subscription created.

451 The schema declaration for the <Subscription> element is derived from the correspondingly named type defined in
452 [[LibertySUBS](#)]. The schema declaration is shown below:

```
453  
454 <xs:element name="Subscription" type="subs:SubscriptionType"/>  
455
```

456 **3.8.1.1. Selecting Objects**

457 The <Object> element to which the WSC is subscribing for change notifications is specified through the targetting
458 mechanisms of the request message in which the <Subscription> element is embedded, either an <Object>
459 element being created or an existing <Object> being targetted through a <TargetObjectID> element.

460 Consequently, the selection mechanisms provided by the Subscription element itself MUST NOT be used.

461 **3.8.1.2. Triggers**

462 This specification defines no triggers.

463 There MUST be no <Trigger> element present in a subscription as the implied trigger is "on change", where the
464 criteria for such change are implicit from the request message in which the <Subscription> element lies.

465 For instance, when a <Subscription> is used in a <AddEntityRequest> message the implied "change" is
466 that an identity token for the created object becomes available; but when a <Subscription> is used in a
467 <ListMembersRequest> message, the change of interest is the membership of the targetted object.

468 **3.8.1.3. Subscription Start**

469 A WSC MAY use the *starts* attribute to indicate the time at which it desires the subscription be in effect.

470 **3.8.1.4. Subscription Aggregation**

471 This specification defines no mechanisms by which notifications can be aggregated.

472 **3.8.1.5. Subscription Expiration**

473 A WSC MUST specify a time at which a subscription expires using the *expires* attribute.

474 A PS provider MAY choose to reject a subscription request if the *expires* attribute is unacceptable. If it does so, the PS
475 provider SHOULD return a second level status code of *InvalidExpires* attribute.

476 **3.8.1.6. Subscription Querying and Management**

477 This specification defines no mechanisms by which an existing subscription can be managed, (e.g. queried, modified,
478 or deleted) beyond those defined in [[LibertySUBS](#)].

479 **3.8.1.7. Including Data**

480 A WSC MAY use the *includeData* attribute to indicate that it wishes to only receive notifications that the object of
481 interest has changed rather than the actual changed <Object>.

482 If no *includeData* attribute is specified, the default value is "yes", e.g. the changed <Object> MUST be returned.

483 **3.8.2. Notify and NotifyResponse Messages**

484 If and when the <Object> corresponding to a <Subscription> element changes, the PS provider MUST use a
485 <Notify> message to indicate this to the WSC.

486 After receiving a <Notify> message from a PS provider, a WSC MAY acknowledge this with a <NotifyResponse>
487 message.

488 The schema declarations for the <Notify> and <NotifyResponse> messages are derived from the correspondingly
489 named types defined in [[LibertySUBS](#)]. The schema declarations are shown below:

```
490  
491 <xs:element name="Notify" type="NotifyType" />  
492  
493 <xs:complexType name="NotifyType">  
494 <xs:complexContent>  
495 <xs:extension base="RequestAbstractType">  
496 <xs:sequence>  
497 <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />  
498 </xs:sequence>  
499 <xs:attributeGroup ref="subs:NotifyAttributeGroup" />  
500 </xs:extension>  
501 </xs:complexContent>  
502 </xs:complexType>  
503  
504 <xs:element name="NotifyResponse" type="subs:NotifyResponseType" />  
505
```

506 3.8.3. <Notification> Element

507 The PS provider MAY send the changed <Object> to the subscriber within the <ItemData> element. If the
508 <ItemData> element is empty, the PS provider is indicating only that the corresponding <Object> has changed.

509 The value of the SubscriptionID attribute on the <Notification> element MUST match that of the
510 SubscriptionID attribute on the <Subscription> element corresponding to which the <Notification>
511 is being sent.

512 The schema declaration for the <Notification> element is derived from the correspondingly named type defined in
513 [[LibertySUBS](#)]. The schema declaration is shown below:

```
514  
515 <xs:element name="Notification" type="NotificationType" />  
516  
517 <xs:complexType name="NotificationType">  
518 <xs:complexContent>  
519 <xs:extension base="subs:NotificationType">  
520 <xs:sequence>  
521 <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded" />  
522 </xs:sequence>  
523 </xs:extension>  
524 </xs:complexContent>  
525 </xs:complexType>  
526
```

527 The schema declaration for the <Object> element is shown below:

```
528  
529 <xs:element name="ItemData" type="ItemDataType" />  
530  
531 <xs:complexType name="ItemDataType">  
532 <xs:sequence>  
533 <xs:element ref="Object" />  
534 </xs:sequence>  
535 </xs:complexType>  
536
```

537 3.9. Adding an Entity

538 A WSC indicates to the PS provider that it wishes a user `Object` to be created by sending an `<AddEntityRequest>`
539 message. The `Object` being created **MUST** be a `urn:liberty:ps:entity` `Object`.

540 **3.9.1. wsa:Action Values**

541 `<AddEntityRequest>` messages **MUST** include a `<wsa:Action>` SOAP header with the value of
542 "urn:liberty:ps:2006-08:AddEntityRequest". `<AddEntityResponse>` messages **MUST** include a `<wsa:Action>`
543 SOAP header with the value of "urn:liberty:ps:2006-08:AddEntityResponse".

544 **3.9.2. AddEntityRequest Message**

545 A WSC uses the `<AddEntityRequest>` message to request that a specified `<Object>` be created.

546 The `<AddEntityRequest>` **MUST NOT** be used to add a new `Object` to an existing `Object`, nor to create two
547 nested `Objects`.

548 The presence of a `<Subscription>` element indicates to the PS provider that the WSC desires that the PS provider
549 return to it (when later possible) an identity token for the invited user within a `<Notify>` message - this possible after
550 a federation has been established between the PS provider and the appropriate IDP. If no `<Subscription>` element
551 is present, the WSC is indicating that the PS provider need not return an identity token through this mechanism.

552 A PS provider can also itself use the `<AddEntityRequest>` message to request that an `Object` be added to a PS list.
553 Typically, this will happen to ensure bilateral PS lists, e.g. if a user is added to a friend's PS, then the friend will be
554 added to the user's PS.

555 The `<AddEntityRequest>` message has the complex type `AddEntityRequestType`, which extends
556 `RequestAbstractType` and adds the following elements:

557 `<Object>` **[Required]** The `<Object>` element is used to convey the target user `Object` being added.

558 `<PStoSPRedirectURL>` **[Optional]** The `<PStoSPRedirectURL>` element is used to convey the URL to which
559 a PS provider will redirect the invited users after federating their IDP account to the PS
560 provider.

561 `<CreatePSObject>` **[Optional]** The `<CreatePSObject>` element allows a WSC to indicate that it desires the PS
562 provider create (or verify the existence of) an `Object` for the inviting user at the PS provider
563 of the invited user (i.e. create or verify the reciprocal relationship).

564 `<Subscription>` **[Optional]** The `<Subscription>` element is used to indicate to the PS provider that the WSC
565 desires that the PS provider return to it (when later possible) an identity token for the invited
566 user.

567 `<sec:TokenPolicy>` **[Optional]** The `<sec:TokenPolicy>` element is used as a container for the WSC's policy
568 requirements of any identity token that it might ask to be returned.

569 The schema declaration for the <AddEntityRequest> message is shown below.

```

570
571 <!-- Declaration of AddEntityRequest element -->
572 <xs:element name="AddEntityRequest" type="AddEntityRequestType"/>
573 <!-- Definition of AddEntityRequestType -->
574 <xs:complexType name="AddEntityRequestType">
575   <xs:complexContent>
576     <xs:extension base="RequestAbstractType">
577       <xs:sequence>
578         <xs:element ref="Object"/>
579         <xs:element ref="PStoSPRedirectURL" minOccurs="0"/>
580         <xs:element ref="CreatePSObject" minOccurs="0"/>
581         <xs:element ref="Subscription" minOccurs="0"/>
582         <xs:element ref="TokenPolicy" minOccurs="0"/>
583       </xs:sequence>
584     </xs:extension>
585   </xs:complexContent>
586 </xs:complexType>
587

```

588 The following is an example of an <AddEntityRequest> message used to create an Object for a user. The WSC
 589 is not requesting that an identity token for the newly created user Object be returned. If it desired this, it would
 590 include a <Subscription> element and, optionally, a <sec:TokenPolicy> element indicating its requirements of
 591 the returned identity token.

```

592
593 <AddEntityRequest>
594   <Object NodeType="urn:liberty:ps:entity">
595     <DisplayName>Alison</DisplayName>
596   </Object>
597   <PStoSPRedirectURL>some SP URL</PStoSPRedirectURL>
598 </AddEntityRequest>
599

```

600 3.9.3. AddEntityResponse Message

601 A PS provider responds to an <AddEntityRequest> message with an <AddEntityResponse> message containing
 602 the newly created <Object> element.

603 The <AddEntityResponse> message has the complex type **AddEntityResponseType**, which extends
 604 **ResponseAbstractType** and adds the following elements:

605 <Object> **[Optional]** The <Object> element is used to convey the Object element just created at the PS provider.

606 <SptoPSRedirectURL> **[Optional]** The <SptoPSRedirectURL> element is used to convey the URL to which
 607 the PS provider desires the invited user be sent if and when they respond to the invitation that
 608 the SP will compose and deliver.

609 <QueryString> **[Optional]** The <QueryString> element is used to convey a SAML artifact (and optional
 610 relay state info) to the invited user, which they can then present to an appropriate provider
 611 (e.g., to an identity provider of the invited user through a cut-and-paste operation into some
 612 HTML form). When a provider receives the artifact, after obtaining consent from the invited
 613 user, it can then use the SAML <samlp:ArtifactResolve> message to dereference the
 614 <QueryString> into a relevant message.

615 This element is defined to offer better protection against identity theft attacks during the
 616 invitation process. See [Section 4.1](#) for more detail.

617 If an issuer of the artifact intends to exchange SAML messages over SAML proto-
 618 col[[SAMLCore2](#)], the value of the artifact itself, and optional relay state information con-
 619 veyed in the <QueryString> element, **MUST** satisfy the formatting and encoding require-
 620 ments of the SAML Artifact Binding (see [\[SAMLBind2\]](#)) as specified by the URI identifier
 621 *urn:oasis:names:tc:SAML:2.0:artifact-04*.

622 The schema declaration for the <AddEntityResponse> message is shown below.

```
623
624 <!-- Declaration of AddEntityResponse element -->
625 <xs:element name="AddEntityResponse" type="AddEntityResponseType" />
626 <!-- Definition of AddEntityResponseType -->
627 <xs:complexType name="AddEntityResponseType">
628   <xs:complexContent>
629     <xs:extension base="ResponseAbstractType">
630       <xs:sequence>
631         <xs:element ref="Object" minOccurs="0"/>
632         <xs:element ref="SPtoPSRedirectURL" minOccurs="0"/>
633         <xs:element ref="QueryString" minOccurs="0"/>
634       </xs:sequence>
635     </xs:extension>
636   </xs:complexContent>
637 </xs:complexType>
638
```

639 The following is an example of an <AddEntityResponse> to the <AddEntityRequest> message above. The PS
 640 provider is responding that the request that Alison be added was successful and returns the created <Object> element
 641 and <SPtoPSRedirectURL> element to which the PS provider desires the invited user be sent if and when they
 642 respond to the invitation that the SP will compose and deliver.

```
643
644 <AddEntityResponse>
645   <Status code="OK" />
646   <Object NodeType="urn:liberty:ps:entity">
647     <ObjectID>https://ps.com/kudfhgs</ObjectID>
648     <DisplayName>Alison</DisplayName>
649   </Object>
650   <SPtoPSRedirectURL>some PS URL</SPtoPSRedirectURL>
651 </AddEntityResponse>
652
```

653 3.9.4. Processing Rules

654 The WSC:

- 655 • **MUST** include an <Object> element within the <AddEntityRequest> message.
- 656 • **MUST** include a *NodeType* attribute on the <Object> element with a value of *urn:liberty:ps:entity*.

-
- 657 • MUST include at least one <DisplayName> element for the invited user within the <Object> element. This
658 element contains the friendly name that the user desires be used for the created *urn:liberty:ps:entity* Object.
- 659 • MAY include a <PStoSPRedirectURL> element.
- 660 • MAY, if it desires that the PS provider create (if not already existing) an Object for the inviting user at the PS
661 provider of the invited user, include a <CreatePSObject> element.
- 662 • MAY, if it desires that an identity token be returned to it through a subsequent <Notification> message, include
663 a <Subscription> element. The presence of a <Subscription> element indicates to the PS provider that the
664 SP desires that the PS provider return to it (when later possible) an identity token for the invited user within a
665 <Notification> in a <Notify> message - this is possible after a federation has been established between the
666 PS provider and the appropriate IDP. If no <Subscription> element is present, the SP is indicating that the PS
667 provider need not return an identity token through this mechanism.
- 668 If the WSC includes a <Subscription> element, it MAY specify its requirements of the eventually re-
669 turned identity token by including a <sec:TokenPolicy> element. The WSC SHOULD NOT include a
670 <sec:TokenPolicy> element unless also including a <Subscription> element.
- 671 In responding to an <AddEntityRequest> message, the PS provider:
- 672 • SHOULD include either or both of a <SPtoPSRedirectURL> or <QueryString> element in the
673 <AddEntityResponse> message returned to the calling SP.
- 674 • MUST be prepared for the invited user to, at some point in the future, visit the URL provided in any specified
675 <SPtoPSRedirectURL> element. As it may be some time before the invited user does respond, the PS provider
676 SHOULD store such a URL for a reasonable length of time.
- 677 MUST, if and when the invited user does respond to the URL specified by the <SPtoPSRedirectURL> element,
678 endeavor to establish a federated identifier for that user with the appropriate identity provider (see [Section 4](#))
- 679 SHOULD, if and when such a federated identifier is established, send an <ims:IdentityMappingRequest>
680 message to that IDP requesting a long-lived identity token (targeted at itself as the provider) for the user for which
681 the federated identifier was just established.
- 682 • SHOULD, if the <AddEntityRequest> message contained a <CreatePSObject> element, ensure that there be
683 an object for the inviting user in the PS of the invited user.
- 684 It may be the case that the inviting user is in the PS of the invited user as a result of a prior invitation sequence
685 initiated 'from the other side'. The PS of the inviting user MUST ensure that no duplicate object be added.
- 686 SHOULD, in order to determine whether an object for the inviting user already exists, query the members of the
687 PS of the invited user using the <ListMembersRequest> message.
- 688 SHOULD, if there is no existing object for the inviting user, request that an object be created with either the
689 <AddEntityRequest> or <AddKnownEntityRequest> messages.
- 690 SHOULD, if sending a <AddKnownEntityRequest> message for the addition, include a <sec:Token> element
691 carrying a token for the inviting user.
- 692 • SHOULD, if the <AddEntityRequest> message contained a <Subscription> element, send an
693 <ims:IdentityMappingRequest> message to that IDP requesting an identity token for the user for
694 which the federated identifier was established but in the namespace of the requesting SP.
- 695 This <ims:IdentityMappingRequest> message to the IDP MUST include any policy directives present in the
696 <AddEntityRequest>.

697 • **SHOULD**, if the `<AddEntityRequest>` message contained a `<Subscription>` element and the
698 `<ims:IdentityMappingRequest>` message to the IDP resulted in an identity token for the user being
699 returned, forward on this identity token to the SP within a `<Notification>` element in a `<Notify>` message
700 corresponding to the original `<Subscription>` element.

701 **3.10. Adding a Known Entity**

702 If a WSC knows an identifier for a user at some identity provider, it can provide this to the PS provider in an
703 `<AddKnownEntityRequest>` message. This known identifier can act as a *bootstrap* for the establishment of the
704 necessary federations. For instance, if the inviting user provides an email address for the invited user, this address
705 may allow the identity provider for that user to be ascertained, thereby obviating the need to ask the user for this
706 information.

707 A WSC indicates to the PS provider that it wishes a known user `Object` to be created by sending an
708 `<AddKnownEntityRequest>` message. The `Object` being created **MUST** be a `urn:liberty:ps:entity` `Object`.
709 The `<AddKnownEntityRequest>` message carries the known identifier for the relevant user within.

710 As for the `<AddKnownEntity>` message, the presence of a `<Subscription>` element indicates to the PS provider
711 that the WSC desires that the PS provider return to it (when later possible) an identity token for the invited user within
712 a `<Notification>` element in a `<Notify>` message - this possible after a federation has been established between
713 the PS provider and the appropriate IDP. If no `<Subscription>` element is present, the WSC is indicating that the
714 PS provider need not return an identity token through this mechanism.

715 **3.10.1. wsa:Action Values**

716 `<AddKnownEntityRequest>` messages **MUST** include a `<wsa:Action>` SOAP header with the value of
717 "urn:liberty:ps:2006-08:AddKnownEntityRequest". `<AddKnownEntityResponse>` messages **MUST** include a
718 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:AddKnownEntityResponse".

719 **3.10.2. AddKnownEntityRequest Message**

720 A WSC uses the `<AddKnownEntityRequest>` message to request that a specified `<Object>` be created for the
721 known user.

722 The `<AddKnownEntityRequest>` **MUST NOT** be used to add a new `Object` to an existing `Object`, nor to create
723 two nested `Objects`.

724 The `<AddKnownEntityRequest>` **MUST** include an appropriate identity token for the target `Object` being created.
725 The `<sec:Token>` element will carry the known identifier for the user.

726 The `<AddKnownEntityRequest>` message has the complex type **AddKnownEntityRequestType**, which extends
727 **RequestAbstractType** and adds the following elements:

728 `<Object>` **[Required]** The `<Object>` element is used to convey the target `Object` being added.

729 `<sec:Token>` **[Required]** The `<sec:Token>` element is used to convey an identity token for the target user
730 `Object` being created.

731 `<CreatePSObject>` **[Optional]** The `<CreatePSObject>` element is used as a directive with which a WSC
732 indicates that it desires a PS provider create (or verify the existence of) an `Object` for the
733 inviting user at the PS provider of the invited user.

734 `<Subscription>` **[Optional]** The `<Subscription>` element is used to indicate to the PS provider that the WSC
735 desires that the PS provider return to it (when later possible) an identity token for the invited
736 user.

737 <sec:TokenPolicy> **[Optional]** The <sec:TokenPolicy> element is used as a container for WSC's require-
 738 ments to an identity token.

739 The schema declaration for the <AddKnownEntityRequest> message is shown below.

```

740
741 <!-- Declaration of AddKnownEntityRequest element -->
742 <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType"/>
743 <!-- Definition of AddKnownEntityRequestType -->
744 <xs:complexType name="AddKnownEntityRequestType">
745   <xs:complexContent>
746     <xs:extension base="RequestAbstractType">
747       <xs:sequence>
748         <xs:element ref="Object"/>
749         <xs:element ref="sec:Token"/>
750         <xs:element ref="CreatePSObject" minOccurs="0"/>
751         <xs:element ref="Subscription" minOccurs="0"/>
752         <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
753       </xs:sequence>
754     </xs:extension>
755   </xs:complexContent>
756 </xs:complexType>
757
```

758 The following is an example of an <AddKnownEntityRequest> message used to create an Object for a user.

```

759
760 <AddKnownEntityRequest>
761   <Object NodeType="urn:liberty:ps:entity">
762     <DisplayName>Bob</DisplayName>
763   </Object>
764   <sec:Token>
765     <saml:Assertion>
766       <saml:Subject>
767         <saml:NameID></saml:NameID>
768       </saml:Subject>
769     </saml:Assertion>
770   </sec:Token>
771 </AddKnownEntityRequest>
772
```

773 3.10.3. AddKnownEntityResponse Message

774 A PS provider responds to an <AddKnownEntityRequest> message with an <AddKnownEntityResponse>
 775 message, in which the PS provider MAY contain the newly created <Object> element.

776 The <AddKnownEntityResponse> message has the complex type **AddKnownEntityResponseType**, which ex-
 777 tends **ResponseAbstractType** and adds the following elements:

778 <Object> **[Optional]** The <Object> element is used to convey the Object element just created at the PS provider.

779 <SptoPSredirectURL> **[Optional]** The <SptoPSredirectURL> element is used to convey the URL to which
 780 the PS provider desires the invited user be sent if and when they respond to the invitation that
 781 the SP will compose and deliver.

782 <QueryString> **[Optional]** The <QueryString> element is used to convey a SAML artifact (and optional
783 relay state info) to the invited user, which they can then present to an appropriate provider
784 (e.g., to an identity provider of the invited user through a cut-and-paste operation into some
785 HTML form). When a provider receives the artifact, after obtaining consent from the invited
786 user, it can then use the SAML <samlp:ArtifactResolve> message to dereference the
787 <QueryString> into a relevant message.

788 This element is defined to offer better protection against identity theft attacks during the
789 invitation process. See [Section 4.1](#) for more detail.

790 If the issuer of the artifact intends to exchange SAML messages over SAML proto-
791 col [[SAMLCore2](#)], the value of the artifact itself, and optional information conveyed in
792 the <QueryString> element, **MUST** satisfy the formatting and encoding requirements
793 of the SAML Artifact Binding (see [[SAMLBind2](#)]) as specified by the URI identifier
794 *urn:oasis:names:tc:SAML:2.0:artifact-04*.

795 The schema declaration for the <AddKnownEntityResponse> message is shown below.

```
796
797 <!-- Declaration of AddKnownEntityResponse element -->
798 <xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType"/>
799 <!-- Definition of AddKnownEntityResponseType -->
800 <xs:complexType name="AddKnownEntityResponseType">
801   <xs:complexContent>
802     <xs:extension base="ResponseAbstractType">
803       <xs:sequence>
804         <xs:element ref="Object" minOccurs="0"/>
805         <xs:element ref="SPToPSRedirectURL" minOccurs="0" maxOccurs="1"/>
806         <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
807       </xs:sequence>
808     </xs:extension>
809   </xs:complexContent>
810 </xs:complexType>
811
```

812 The following is an example of an <AddKnownEntityResponse> to the <AddKnownEntityRequest> message
813 above. The PS provider is responding that the request that Bob be added was successful and returns the created
814 <Object> element.

```
815
816 <AddKnownEntityResponse>
817   <Status code="OK"/>
818   <Object NodeType="urn:liberty:ps:entity">
819     <ObjectID>https://ps.com/lafnervf</ObjectID>
820     <DisplayName>Bob</DisplayName>
821   </Object>
822 </AddKnownEntityResponse>
823
```

824 3.10.4. Processing Rules

825 The WSC:

- 826 • **MUST** include an <Object> element within the <AddKnownEntityRequest> message.
- 827 • **MUST** include a *NodeType* attribute on the <Object> element with a value of *urn:liberty:ps:entity*.
- 828 • **MUST** include a <Token> element within the <AddKnownEntityRequest> message.

- 829 • MUST, if a SAML <Assertion> is used as the identity token format, specify the known identifier in that
830 assertion's <Subject> element.
- 831 • MUST include at least one <DisplayName> for the invited user within the <Object> element. This element
832 contains the friendly name that the user desires be used for the created user Object.
- 833 • MAY, if it desires that the PS provider create (or verify the existence of) an Object for the inviting user at the PS
834 provider of the invited user, include a <CreatePSObject> element.
- 835 • MAY, if it desires that an identity token be returned to it through a subsequent <Notification> message, include
836 a <Subscription> element. The presence of a <Subscription> element indicates to the PS provider that the
837 SP desires that the PS provider return to it (when later possible) an identity token for the invited user within a
838 <Notification> message - this possible after a federation has been established between the PS provider and the
839 appropriate IDP. If no <Subscription> element is present, the SP is indicating that the PS provider need not
840 return an identity token through this mechanism.
- 841 In responding to an <AddKnownEntityRequest> message, the PS provider:
- 842 • MAY include either or both of a <SPToPSRedirectURL> or <QueryString> element in the
843 <AddKnownEntityResponse> message returned to the calling SP.
- 844 • SHOULD, if the <AddKnownEntityRequest> message contained a <Subscription> element, send a
845 <ims:IdentityMappingRequest> message to that IDP requesting an identity token for the user for which the
846 federated identifier was established but in the namespace of the requesting SP.
- 847 This <ims:IdentityMappingRequest> message to the IDP MUST include any policy directives present in the
848 <AddKnownEntityRequest>.
- 849 • SHOULD, if the <AddKnownEntityRequest> message contained a <Subscription> element and the
850 <ims:IdentityMappingRequest> message to the IDP resulted in an identity token for the user being returned,
851 forward on this identity token to the SP within a <Notification> message corresponding to the original
852 <Subscription> element.
- 853 • SHOULD, if the <AddKnownEntityRequest> message contained a <CreatePSObject> element, ensure that
854 there be an object for the inviting user in the PS of the invited user.
- 855 It may be the case that the inviting user is in the PS of the invited user as a result of a prior invitation sequence
856 initiated 'from the other side'. The PS of the inviting user MUST ensure that no duplicate object be added.
- 857 SHOULD, in order to determine whether an object for the inviting user already exists, query the members of the
858 PS of the invited user using the <ListMembersRequest> message.
- 859 SHOULD, if there is no existing object for the inviting user, request that an object be created with either the
860 <AddEntityRequest> or <AddKnownEntityRequest> messages.
- 861 SHOULD, if sending a <AddKnownEntityRequest> message for the addition, include a <sec:Token> element
862 carrying a token for the inviting user.

863 3.11. Removing an Entity

864 A WSC indicates to the PS provider that it wishes a user Object to be completely removed from the PS resource
865 by sending a <RemoveEntityRequest> message. The Object being removed MUST be a *urn:liberty:ps:entity*
866 Object.

867 3.11.1. wsa:Action Values

868 <RemoveEntityRequest> messages MUST include a <wsa:Action> SOAP header with the value of
869 "urn:liberty:ps:2006-08:RemoveEntityRequest". <RemoveEntityResponse> messages MUST include a
870 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:RemoveEntityResponse".

871 3.11.2. RemoveEntityRequest Message

872 A WSC uses the <RemoveEntityRequest> message to request that a user Object corresponding to the value of the
873 specified <TargetObjectID> element be removed.

874 The <RemoveEntityRequest> message is used to completely remove a user Object from the PS resource. To
875 simply remove a child user <Object> element from some parent group <Object>, the <RemoveEntityRequest>
876 message MUST NOT be used, but rather a <RemoveFromCollectionRequest> message with the parent Object's
877 ObjectID specified in the <TargetObjectID> element, MUST be used (see [Section 3.15](#) for more details).

878 The <RemoveEntityRequest> message has the complex type **RemoveEntityRequestType**, which extends
879 **RequestAbstractType** and adds the following element:

880 <TargetObjectID> **[Required]** The <TargetObjectID> element is used to convey one or more ObjectID's of
881 the target user Objects being removed.

882 The schema declaration for the <RemoveEntityRequest> message is shown below.

```
883  
884 <!-- Declaration of RemoveEntityRequest element -->  
885 <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType"/>  
886 <!-- Definition of RemoveEntityRequestType -->  
887 <xs:complexType name="RemoveEntityRequestType">  
888   <xs:complexContent>  
889     <xs:extension base="RequestAbstractType">  
890       <xs:sequence>  
891         <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>  
892       </xs:sequence>  
893     </xs:extension>  
894   </xs:complexContent>  
895 </xs:complexType>  
896
```

897 The following is an example of a <RemoveEntityRequest> message used to remove an Object for a user.

```
898  
899 <RemoveEntityRequest>  
900   <TargetObjectID>https://ps.com/lafnervf</TargetObjectID>  
901 </RemoveEntityRequest>  
902
```

903 3.11.3. RemoveEntityResponse Message

904 A PS provider responds to a <RemoveEntityRequest> message with a <RemoveEntityResponse> element. A
905 PS provider removes the specified <Object> and responds with a status of this process based on the processing rules
906 described in section [Section 3.11.4](#).

907 The <RemoveEntityResponse> message has the type of **ResponseAbstractType**.

908 The schema declaration for the <RemoveEntityResponse> message is shown below.

```
909  
910 <!-- Declaration of RemoveEntityResponse element -->  
911 <xs:element name="RemoveEntityResponse" type="ResponseAbstractType"/>  
912
```

913 The following is an example of a <RemoveEntityResponse> to the <RemoveEntityRequest> message above.
914 The PS provider is responding that the request that a specified Object be removed was successful.

```
915  
916 <RemoveEntityResponse>  
917   <Status code="OK"/>  
918 </RemoveEntityResponse>  
919
```

920 **3.11.4. Processing Rules**

921 The WSC:

- 922 • MUST ensure that the targeted <Object> has a `NodeType` attribute with a value of `urn:liberty:ps:entity`.

923 The PS provider:

- 924 • MAY cancel any existing federated identifier with the relevant IDP for that user being removed.

925 **3.12. Adding a Collection**

926 A WSC indicates to the PS provider that it wishes a group Object to be created by sending an
927 <AddCollectionRequest> message. The Object being created MUST be a `urn:liberty:ps:collection` Object.

928 **3.12.1. wsa:Action Values**

929 <AddCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of
930 "urn:liberty:ps:2006-08:AddCollectionRequest". <AddCollectionResponse> messages MUST include a
931 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:AddCollectionResponse".

932 **3.12.2. AddCollectionRequest Message**

933 A WSC uses the <AddCollectionRequest> message to request that a specified group <Object> be created.

934 The <AddCollectionRequest> MUST NOT be used to add a new group Object to an existing group Object.
935 Instead the <AddToCollectionRequest> MUST be used (see [Section 3.14](#))

936 The <AddCollectionRequest> message has the complex type **AddCollectionRequestType**, which extends
937 **RequestAbstractType** and adds the following element:

938 <Object> **[Required]** The <Object> element is used to convey the target group Object being added.

939 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
940 desires that the PS provider send a notification if and when the group object being added
941 changes.

942 The schema declaration for the <AddCollectionRequest> message is shown below.

```

943
944 <!-- Declaration of AddCollectionRequest element -->
945 <xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
946 <!-- Definition of AddCollectionRequestType -->
947 <xs:complexType name="AddCollectionRequestType">
948   <xs:complexContent>
949     <xs:extension base="RequestAbstractType">
950       <xs:sequence>
951         <xs:element ref="Object"/>
952         <xs:element ref="Subscription" minOccurs="0"/>
953       </xs:sequence>
954     </xs:extension>
955   </xs:complexContent>
956 </xs:complexType>
957
```

958 The following is an example of an <AddCollectionRequest> message used to create an Object for a group.

```

959
960 <AddCollectionRequest>
961   <Object NodeType="urn:liberty:ps:collection">
962     <DisplayName>Soccer Team</DisplayName>
963   </Object>
964 </AddCollectionRequest>
965
```

966 3.12.3. AddCollectionResponse Message

967 A PS provider responds to an <AddCollectionRequest> message with an <AddCollectionResponse> message
968 containing the newly created <Object> element.

969 The <AddCollectionResponse> message has the complex type **AddCollectionResponseType**, which extends
970 **ResponseAbstractType** and adds the following element:

971 <Object> **[Optional]** The <Object> element is used to convey the Object element just created at the PS provider.

972 The schema declaration for the <AddCollectionResponse> message is shown below.

```

973
974 <!-- Declaration of AddCollectionResponse element -->
975 <xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>
976 <!-- Definition of AddCollectionResponseType -->
977 <xs:complexType name="AddCollectionResponseType">
978   <xs:complexContent>
979     <xs:extension base="ResponseAbstractType">
980       <xs:sequence>
981         <xs:element ref="Object" minOccurs="0"/>
982       </xs:sequence>
983     </xs:extension>
984   </xs:complexContent>
985 </xs:complexType>
986
```

987 The following is an example of an <AddCollectionResponse> to the <AddCollectionRequest> message above.
988 The PS provider is responding that the request that the Soccer Team be added was successful and returns the created
989 <Object> element.

```

990
991 <AddCollectionResponse>
992   <Status code="OK"/>
993   <Object NodeType="urn:liberty:ps:collection">
```

```
994     <ObjectID>https://ps.com/roqlsfof</ObjectID>
995     <DisplayName>Soccer Team</DisplayName>
996   </Object>
997 </AddCollectionResponse>
998
```

999 3.12.4. Processing Rules

1000 The WSC:

- 1001 • MUST include an `<Object>` element within the `<AddCollectionRequest>` message.
- 1002 • MUST include a `NodeType` attribute on the `<Object>` element with a value of `urn:liberty:ps:collection`.
- 1003 • MUST include at least one `<DisplayName>` element for a group within the `<Object>`. This element contains the
1004 friendly name that the user desires be used for the created group Object.
- 1005 In responding to an `<AddCollectionRequest>` message, the PS provider:
 - 1006 • MUST return an `<Object>` element within the `<AddCollectionResponse>` message with the same
1007 `<DisplayName>` as specified on the `<AddCollectionRequest>`.
 - 1008 • MUST include an `<ObjectID>` element for the newly created group Object.

1009 3.13. Removing a Collection

1010 A WSC indicates to the PS provider that it wishes a group Object to be removed by sending a
1011 `<RemoveCollectionRequest>` message. The Object being removed MUST be a `urn:liberty:ps:collection`
1012 Object.

1013 3.13.1. wsa:Action Values

1014 `<RemoveCollectionRequest>` messages MUST include a `<wsa:Action>` SOAP header with the value of
1015 "urn:liberty:ps:2006-08:RemoveCollectionRequest". `<RemoveCollectionResponse>` messages MUST include a
1016 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:RemoveCollectionResponse".

1017 3.13.2. RemoveCollectionRequest Message

1018 A WSC uses the `<RemoveCollectionRequest>` message to request that a group Object corresponding to the value
1019 of the specified `<TargetObjectID>` be removed.

1020 The `<RemoveCollectionRequest>` message is used to completely remove a group Object from the PS
1021 resource. To simply remove a child group `<Object>` element from some parent group `<Object>`, the
1022 `<RemoveCollectionRequest>` message MUST NOT be used, but rather a `<RemoveFromCollectionRequest>`
1023 with the parent Object's ObjectID specified in the `TargetObjectID` element, MUST be used (see [Section 3.15](#)
1024 for more details).

1025 The `<RemoveCollectionRequest>` message does not result in the removal of any child `<Object>` elements unless
1026 they are explicitly identified through separate `<TargetObjectID>` elements.

1027 The `<RemoveCollectionRequest>` message has the complex type `RemoveCollectionRequestType`, which
1028 extends `RequestAbstractType` and adds the following element:

1029 `<TargetObjectID>` **[Required]** The `<TargetObjectID>` element specifies the ObjectID of the targeted group
1030 Objects being removed.

1031 The schema declaration for the <RemoveCollectionRequest> message is shown below.

```
1032
1033 <!-- Declaration of RemoveCollectionRequest element -->
1034 <xs:element name="RemoveCollectionRequest" type="RemoveCollectionRequestType"/>
1035 <!-- Definition of RemoveCollectionRequestType -->
1036 <xs:complexType name="RemoveCollectionRequestType" >
1037   <xs:complexContent>
1038     <xs:extension base="RequestAbstractType">
1039       <xs:sequence>
1040         <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
1041       </xs:sequence>
1042     </xs:extension>
1043   </xs:complexContent>
1044 </xs:complexType>
1045
```

1046 The following is an example of a <RemoveCollectionRequest> message used to remove an Object for a group.

```
1047
1048 <RemoveCollectionRequest>
1049   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
1050 </RemoveCollectionRequest>
1051
```

1052 3.13.3. RemoveCollectionResponse Message

1053 A PS provider responds to a <RemoveCollectionRequest> message with a <RemoveCollectionResponse>
1054 element. A PS provider removes the specified <Object> and responds a status of this process based on the processing
1055 rules described in [Section 3.11.4](#).

1056 The <RemoveCollectionResponse> message has the type of **ResponseAbstractType**

1057 The schema declaration for the <RemoveCollectionResponse> message is shown below.

```
1058
1059 <!-- Declaration of RemoveCollectionResponse element -->
1060 <xs:element name="RemoveCollectionResponse" type="ResponseAbstractType"/>
1061
```

1062 The following is an example of a <RemoveCollectionResponse> to the <RemoveCollectionRequest> message
1063 above. The PS provider is responding that the request that a specified Object be removed was successful.

```
1064
1065 <RemoveCollectionResponse>
1066   <Status code="OK"/>
1067 </RemoveCollectionResponse>
1068
```

1069 3.13.4. Processing Rules

1070 The WSC:

- 1071 • MUST include a `NodeType` attribute on the <Object> element with a value of `urn:liberty:ps:collection`.

1072 The PS provider:

- 1073 • MUST remove the specified <Object> from the PS list.

1074 • **MUST NOT**, if the specified group <Object> has one or more child <Object> elements, remove any the child
1075 Objects unless those child <Object>s are explicitly specified by their own <ObjectID> values in separate
1076 <TargetObjectID> elements.

1077 **3.14. Adding to a Collection**

1078 A WSC uses the <AddToCollectionRequest> message to request that child Object elements be added to
1079 an existing group Object. Both user and group Objects can be added to a parent group Object with the
1080 <AddToCollectionRequest> message.

1081 **3.14.1. wsa:Action Values**

1082 <AddToCollectionRequest> messages **MUST** include a <wsa:Action> SOAP header with the value of
1083 "urn:liberty:ps:2006-08:AddToCollectionRequest". <AddToCollectionResponse> messages **MUST** include a
1084 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:AddToCollectionResponse".

1085 **3.14.2. AddToCollectionRequest Message**

1086 The target parent group Object to which child Objects are to be added is indicated by the value of the
1087 <TargetObjectID> element within the <AddToCollectionRequest> element. The child Object's being added
1088 are specified by the values of the <ObjectID> elements within the <AddToCollectionRequest> element.

1089 The <AddToCollectionRequest> message has the complex type **AddToCollectionRequestType**, which ex-
1090 tends **RequestAbstractType** and adds the following elements:

1091 <TargetObjectID> **[Required]** The <TargetObjectID> element is used to convey the ObjectID of the target
1092 group Object to which specified Objects are added.

1093 <ObjectID> **[Required]** The <ObjectID> element is used to convey ObjectID's of the Objects to be added to
1094 the target group Object.

1095 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
1096 desires that the PS provider send a notification if and when the membership of the targeted
1097 group changes.

1098 The schema declaration for the <AddToCollectionRequest> message is shown below.

```
1099 <!-- Declaration of AddToCollectionRequest element -->  
1100 <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>  
1101 <!-- Definition of AddToCollectionRequestType -->  
1102 <xs:complexType name="AddToCollectionRequestType">  
1103   <xs:complexContent>  
1104     <xs:extension base="RequestAbstractType">  
1105       <xs:sequence>  
1106         <xs:element ref="TargetObjectID"/>  
1107         <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>  
1108         <xs:element ref="Subscription" minOccurs="0"/>  
1109       </xs:sequence>  
1110     </xs:extension>  
1111   </xs:complexContent>  
1112 </xs:complexType>  
1113 </xs:complexType>  
1114
```

1115 The following is an example of an <AddToCollectionRequest> message used to add three Objects to the target
1116 group Object.

```
1117 <AddToCollectionRequest>  
1118   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
```

```
1120 <ObjectID>https://ps.com/qaf3eflo</ObjectID>
1121 <ObjectID>https://ps.com/bzpfnrns</ObjectID>
1122 <ObjectID>https://ps.com/nsdflfss</ObjectID>
1123 </AddToCollectionRequest>
1124
```

1125 3.14.3. AddToCollectionResponse Message

1126 A PS provider responds to an <AddToCollectionRequest> message with an <AddToCollectionResponse>
1127 message. The PS provider makes the indicated modification to the specified target <Object> and responds with the
1128 status.

1129 The <AddToCollectionResponse> message has the type of **ResponseAbstractType**

1130 The schema declaration for the <AddToCollectionResponse> message is shown below.

```
1131
1132 <!-- Declaration of AddToCollectionResponse element -->
1133 <xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>
1134
```

1135 The following is an example of an <AddToCollectionResponse> to the <AddToCollectionRequest> message
1136 above. The PS provider is responding that the request that the three Objects be added to the target Object was
1137 successful.

```
1138
1139 <AddToCollectionResponse>
1140   <Status code="OK"/>
1141 </AddToCollectionResponse>
1142
```

1143 3.14.4. Processing Rules

1144 The WSC:

1145 •MUST specify, as a value of the <TargetObjectID>, an ObjectID of the Object that has
1146 *urn:liberty:ps:collection* as a value of Node Type attribute.

1147 The PS provider:

1148 •MUST, if the Object specified by the value of the <TargetObjectID> element has *urn:liberty:ps:entity* as the
1149 value of its Node Type attribute, respond with *Failed* as the code attribute of the top level <lu:Status> element,
1150 and the code attribute of the second level <lu:Status> element MUST be set with the following status code:

1151 • ObjectIsEntity

1152 3.15. Removing from a Collection

1153 A WSC uses the <RemoveFromCollectionRequest> message to request the removal of a child Object element
1154 from a parent group Object. Both user and group Objects can be removed from a parent group Object with the
1155 <RemoveFromCollectionRequest> message.

1156 3.15.1. wsa:Action Values

1157 <RemoveFromCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1158 "urn:liberty:ps:2006-08:RemoveFromCollectionRequest". <RemoveFromCollectionResponse> messages MUST
1159 include a <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:RemoveFromCollectionResponse".

1160 3.15.2. RemoveFromCollectionRequest Message

1161 The target parent group Object from which a child Object is to be removed is indicated by the value of the
1162 <TargetObjectID> element within the <RemoveFromCollectionRequest> message. The child Object being
1163 removed are specified by the value(s) of the <ObjectID> elements within the <RemoveFromCollectionRequest>
1164 element.

1165 The <RemoveFromCollectionRequest> message has the complex type **RemoveFromCollectionRequestType**,
1166 which extends **RequestAbstractType** and adds the following elements:

1167 <TargetObjectID> **[Required]** The <TargetObjectID> element is used to convey the ObjectID of the target
1168 group Object from which specified Objects are to be removed.

1169 <ObjectID> **[Required]** The <ObjectID> element is used to convey ObjectID's of the Objects that would be
1170 removed from the target group Object.

1171 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
1172 desires that the PS provider send a notification if and when the membership of the targeted
1173 group changes.

1174 The schema declaration for the <RemoveFromCollectionRequest> message is shown below.

```
1175  
1176 <!-- Declaration of RemoveFromCollectionRequest element -->  
1177 <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType" />  
1178 <!-- Definition of RemoveFromCollectionRequestType -->  
1179 <xs:complexType name="RemoveFromCollectionRequestType">  
1180   <xs:complexContent>  
1181     <xs:extension base="RequestAbstractType">  
1182       <xs:sequence>  
1183         <xs:element ref="TargetObjectID" />  
1184         <xs:element ref="ObjectID" maxOccurs="unbounded" />  
1185         <xs:element ref="Subscription" minOccurs="0" />  
1186       </xs:sequence>  
1187     </xs:extension>  
1188   </xs:complexContent>  
1189 </xs:complexType>  
1190
```

1191 The following is an example of a <RemoveFromCollectionRequest> message used to remove three Objects from
1192 the target group Object.

```
1193  
1194 <RemoveFromCollectionRequest>  
1195   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>  
1196   <ObjectID>https://ps.com/qaf3eflo</ObjectID>  
1197   <ObjectID>https://ps.com/bzpfnrns</ObjectID>  
1198   <ObjectID>https://ps.com/nsdfllfss</ObjectID>
```

1199 </RemoveFromCollectionRequest>
1200

1201 3.15.3. RemoveFromCollectionResponse Message

1202 A PS provider responds to a <RemoveFromCollectionRequest> message with a <RemoveFromCollectionResponse>
1203 message. The PS provider makes the indicated modification to the specified target Object and responds with the
1204 status.

1205 The <RemoveFromCollectionResponse> message has the type of **ResponseAbstractType**

1206 The schema declaration for the <RemoveFromCollectionResponse> message is shown below.

```
1207  
1208 <!-- Declaration of RemoveFromCollectionResponse element -->  
1209 <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>  
1210
```

1211 The following is an example of a <RemoveFromCollectionResponse> to the <RemoveFromCollectionRequest>
1212 message above. The PS provider is responding that the request that the three objects be removed from the target
1213 Object was successful.

```
1214  
1215 <RemoveFromCollectionResponse>  
1216   <Status code="OK"/>  
1217 </RemoveFromCollectionResponse>  
1218
```

1219 3.15.4. Processing Rules

1220 The WSC:

1221 • **MUST** specify, as a value of the <TargetObjectID>, an ObjectID of the Object that has
1222 *urn:liberty:ps:collection* as a value of Node**Type** attribute.

1223 The PS provider:

1224 • **MUST**, if the Object specified by the value of the <TargetObjectID> element has *urn:liberty:ps:entity* as the
1225 value of its Node**Type** attribute, respond with *Failed* as the code attribute of the top level <lu:Status> element,
1226 and the code attribute of the second level <lu:Status> element **MUST** be set with the following status code:

1227 • ObjectIsEntity

1228 **3.16. Listing Members**

1229 A WSC uses the `<ListMembersRequest>` message to navigate the Object structure of the users
1230 (*urn:liberty:ps:entity* Objects) and groups (*urn:liberty:ps:collection* Objects) that comprise the PS resource.

1231 A WSC can control how Objects are returned by specifying its preferences with the Structured attribute.

1232 If a WSC does not specify a `<TargetObjectID>` element in the `<ListMembersRequest>` message, this is
1233 equivalent to asking for all top-level Object element, i.e. the default targeted Object is the root node.

1234 **3.16.1. wsa:Action Values**

1235 `<ListMembersRequest>` messages MUST include a `<wsa:Action>` SOAP header with the value of
1236 "urn:liberty:ps:2006-08:ListMembersRequest". `<ListMembersResponse>` messages MUST include a
1237 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:ListMembersResponse".

1238 **3.16.2. ListMembersRequest Message**

1239 The WSC indicates to the PS provider the Object of interest by specifying that `<Object>` element's ObjectID in
1240 the `<TargetObjectID>` element. If no `<TargetObjectID>` element is specified in the `<ListMembersRequest>`
1241 message, the PS provider MUST return all the top-level Objects (i.e., Objects that do not have any parent Object.)

1242 The `<ListMembersRequest>` message has the complex type `ListMembersRequestType`, which extends
1243 `RequestAbstractType` and adds the following attributes and elements:

1244 Structured [**Optional**] The Structured allows a WSC to indicate what portion of the Object tree structure it
1245 desires be returned. See [Section 3.16.2.1](#) for more detail.

1246 Count [**Optional**] The Count attribute specifies how many child `<Object>` elements should be returned in a
1247 `<ListMembersResponse>` message. See [Section 3.16.2.2](#) for more detail.

1248 Offset [**Optional**] The Offset attribute specifies from which element to continue, when requesting more data.
1249 See [Section 3.16.2.2](#) for more detail.

1250 `<TargetObjectID>` [**Optional**] The `<TargetObjectID>` element is used to convey an ObjectID of the target
1251 group Object whose child Objects are to be listed.

1252 `<Subscription>` [**Optional**] The `<Subscription>` element is used to indicate to the PS provider that the WSC
1253 desires that the PS provider send a notification if and when the membership of the targeted
1254 group changes.

1255 Interpretation of the membership for which a change `<Notify>` message should be sent will
1256 depend on the value of the Structured attribute on the `<ListMembersRequest>` message.
1257 For instance, if the attribute has a value of "tree", the subscription corresponds to the full
1258 object tree and `<Notify>` messages MUST be sent accordingly.

1259 The schema declaration for the `<ListMembersRequest>` message is shown below.

```

1260
1261 <!-- Declaration of ListMembersRequest element -->
1262 <xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
1263 <!-- Definition of ListMembersRequestType -->
1264 <xs:complexType name="ListMembersRequestType">
1265   <xs:complexContent>
1266     <xs:extension base="RequestAbstractType">
1267       <xs:sequence>
1268         <xs:element ref="TargetObjectID" minOccurs="0"/>
1269         <xs:element ref="Subscription" minOccurs="0"/>
1270       </xs:sequence>
1271       <xs:attribute name="Structured" type="xs:string" use="optional"/>
1272       <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
1273       <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
1274     </xs:extension>
1275   </xs:complexContent>
1276 </xs:complexType>
1277

```

1278 3.16.2.1. Structured Attribute

1279 WSC's may choose to navigate the hierarchal tree structure for a given Object incrementally (i.e. by successive
1280 requests to probe deeper) or to have the complete tree returned. The Structured attribute allows the WSC to indicate
1281 this preference to the PS provider. Additionally, the WSC uses the Structured attribute to indicate whether or not
1282 it desires to see collection Object elements returned or only entity Object elements.

1283 The Structured attribute takes three possible values

1284 *children* the WSC is indicating that it desires only the direct child collection and entity objects.

1285 The default value for the Structured attribute is *children*.

1286 *tree* the WSC is indicating that it desires the full tree structure of the targetted object.

1287 *entities* the WSC is indicating that it desires a flat view of the full tree structure, e.g. one in which any
1288 collection hierarchy is hidden.

1289 3.16.2.2. Count and Offset Attributes

1290 When the specified Object has multiple child Objects, the WSC may desire to be sent the child `<Object>` elements
1291 in smaller sets (i.e., a smaller number of elements at a time). This is achieved by using the Count and Offset attributes
1292 of the `<ListMembersRequest>` element.

1293 The Count attribute defines how many child `<Object>` elements should be returned in a `<ListMembersResponse>`
1294 message. The Count attribute only pertains to the direct child `<Object>` elements of the Object specified in the
1295 `<ListMembersRequest>` message.

1296 The Offset attribute specifies from which element to continue, when requesting for more data. The default value is
1297 zero, which refers to the first child `<Object>` element.

1298 3.16.3. ListMembersResponse Message

1299 A PS provider responds to a `<ListMembersRequest>` message with a `<ListMembersResponse>` message contain-
1300 ing the appropriate set of `<Object>` elements.

1301 The `<ListMembersResponse>` message has the complex type `ListMembersResponseType`, which extends
1302 `ResponseAbstractType` and adds the following element:

1303 <Object> **[Optional]** The <Object> element is used to convey data of zero or more Objects to be listed.

1304 The schema declaration for the <ListMembersResponse> message is shown below.

```

1305
1306 <!-- Declaration of ListMembersResponse element -->
1307 <xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
1308 <!-- Definition of ListMembersResponseType -->
1309 <xs:complexType name="ListMembersResponseType">
1310   <xs:complexContent>
1311     <xs:extension base="ResponseAbstractType">
1312       <xs:sequence>
1313         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
1314       </xs:sequence>
1315     </xs:extension>
1316   </xs:complexContent>
1317 </xs:complexType>
1318
```

1319 3.16.4. Examples

1320 All the examples described in this subsection assume that a PS provider has the following virtual XML document for
 1321 some user's PS list.

```

1322
1323 <Object NodeType="urn:liberty:ps:entity">
1324   <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1325   <DisplayName>Mary</DisplayName>
1326 </Object>
1327 <Object NodeType="urn:liberty:ps:entity">
1328   <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1329   <DisplayName>Bob</DisplayName>
1330 </Object>
1331 <Object NodeType="urn:liberty:ps:entity">
1332   <ObjectID>https://ps.com/reremvls</ObjectID>
1333   <DisplayName>Nick</DisplayName>
1334 </Object>
1335 <Object NodeType="urn:liberty:ps:entity">
1336   <ObjectID>https://ps.com/soijfsfd</ObjectID>
1337   <DisplayName>JoJo</DisplayName>
1338 </Object>
1339 <Object NodeType="urn:liberty:ps:entity">
1340   <ObjectID>https://ps.com/sdosafms</ObjectID>
1341   <DisplayName>Taro</DisplayName>
1342 </Object>
1343 <Object NodeType="urn:liberty:ps:entity">
1344   <ObjectID>https://ps.com/lgsdfsf</ObjectID>
1345   <DisplayName>Hanako</DisplayName>
1346 </Object>
1347 <Object NodeType="urn:liberty:ps:collection">
1348   <ObjectID>https://ps.com/eiruvoie</ObjectID>
1349   <DisplayName>Soccer Team</DisplayName>
1350   <Object NodeType="urn:liberty:ps:collection">
1351     <ObjectID>https://ps.com/nmerflas</ObjectID>
1352     <DisplayName>Starting Members</DisplayName>
1353     <ObjectRef Ref="https://ps.com/lsdjfojd"/>
1354     <ObjectRef Ref="https://ps.com/sijfsfsf"/>
1355   </Object>
1356   <ObjectRef Ref="https://ps.com/reremvls"/>
1357   <ObjectRef Ref="https://ps.com/soijfsfd"/>
1358 </Object>
1359 <Object NodeType="urn:liberty:ps:collection">
1360   <ObjectID>https://ps.com/zxliidfd</ObjectID>
1361   <DisplayName>Family</DisplayName>
1362   <ObjectRef Ref="https://ps.com/sdosafms"/>

```

```
1363 <ObjectRef Ref="https://ps.com/lgsdfsf" />
1364 </Object>
1365
```

1366 The following is an example of a <ListMembersRequest> message by which a WSC requests the list of members
1367 of the "Soccer Team" collection Object. As the WSC specifies Structured="entities", it is indicating that it
1368 desires to have a 'flat' view of that group's Object tree returned, e.g. one in which all collection hierarchy is removed.

```
1369
1370 <ListMembersRequest Structured="entities">
1371   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1372 </ListMembersRequest>
1373
```

1374 The following is an example <ListMembersResponse> message as might be returned to the
1375 <ListMembersRequest> above.

```
1376
1377 <ListMembersResponse>
1378   <Status code="OK" />
1379   <Object NodeType="urn:liberty:ps:entity">
1380     <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1381     <DisplayName>Mary</DisplayName>
1382   </Object>
1383   <Object NodeType="urn:liberty:ps:entity">
1384     <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1385     <DisplayName>Bob</DisplayName>
1386   </Object>
1387   <Object NodeType="urn:liberty:ps:entity">
1388     <ObjectID>https://ps.com/reremvls</ObjectID>
1389     <DisplayName>Nick</DisplayName>
1390   </Object>
1391   <Object NodeType="urn:liberty:ps:entity">
1392     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1393     <DisplayName>JoJo</DisplayName>
1394   </Object>
1395 </ListMembersResponse>
1396
```

1397 As the WSC indicated it desired a flat view, the PS expanded the group Object called "Starting Members" and returns
1398 its two child entity Object elements (for Mary and Bob) instead of the collection Object itself.

1399 Alternatively, the following is a <ListMembersResponse> message as might be returned to a
1400 <ListMembersRequest> message in which the WSC indicated it desired to see the full tree structure by
1401 specifying Structured="tree".

```
1402
1403 <ListMembersResponse>
1404   <Status code="OK" />
1405   <Object NodeType="urn:liberty:ps:collection">
1406     <ObjectID>https://ps.com/nmerflas</ObjectID>
1407     <DisplayName>Starting Members</DisplayName>
1408     <Object NodeType="urn:liberty:ps:entity">
1409       <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1410       <DisplayName>Mary</DisplayName>
1411     </Object>
1412     <Object NodeType="urn:liberty:ps:entity">
1413       <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1414       <DisplayName>Bob</DisplayName>
1415     </Object>
1416   </Object>
1417   <Object NodeType="urn:liberty:ps:entity">
1418     <ObjectID>https://ps.com/reremvls</ObjectID>
1419     <DisplayName>Nick</DisplayName>
1420   </Object>
```

```

1421     <Object NodeType="urn:liberty:ps:entity">
1422         <ObjectID>https://ps.com/soijfsfd</ObjectID>
1423         <DisplayName>JoJo</DisplayName>
1424     </Object>
1425 </ListMembersResponse>
1426

```

1427 The PS returns the full sub-tree for the specified target object including the hierarchy of the "Starting Members" group.

1428 Lastly, the following is an example <ListMembersResponse> message as might be returned to a
1429 <ListMembersRequest> message in which the WSC indicated it desired to see only direct children by spec-
1430 ifying Structured="children".

```

1431
1432 <ListMembersResponse>
1433     <Status code="OK"/>
1434     <Object NodeType="urn:liberty:ps:collection">
1435         <ObjectID>https://ps.com/nmerflas</ObjectID>
1436         <DisplayName>Starting Members</DisplayName>
1437     </Object>
1438     <Object NodeType="urn:liberty:ps:entity">
1439         <ObjectID>https://ps.com/reremvls</ObjectID>
1440         <DisplayName>Nick</DisplayName>
1441     </Object>
1442     <Object NodeType="urn:liberty:ps:entity">
1443         <ObjectID>https://ps.com/soijfsfd</ObjectID>
1444         <DisplayName>JoJo</DisplayName>
1445     </Object>
1446 </ListMembersResponse>
1447

```

1448 3.16.5. Processing Rules

- 1449 • The PS provider SHOULD dereference all <ObjectRef> elements and replace them with the appropriate
1450 <Object> elements before returning.
- 1451 • At any one level of the tree, the PS provider SHOULD remove all duplicate <Object> elements before returning.
- 1452 • If the Structured attribute is set as *tree* the PS provider MUST return all the direct child and descendant
1453 <Object> elements of the specified Object.
- 1454 • If the Structured attribute is not set in the request, the PS provider MUST process it as if it is set as *children*.
- 1455 • If the Structured attribute is set as *entities* the PS provider MUST return all the direct child and descendant
1456 entity <Object> elements of the specified Object. Any collection <Object> elements MUST be removed and
1457 only entity <Object> elements returned.
- 1458 • If the Structured attribute is set as *children* the PS provider MUST return all the direct child collection and
1459 entity <Object> elements of the specified Object.
- 1460 • If a PS provider receives a <ListMembersRequest> message on which the value of <TargetObjectID>
1461 matches that of an <ObjectID> element of a given Object, and if the value of the NodeType attribute of the
1462 matched Object is *urn:liberty:ps:entity*, then the PS provider MUST respond with *Failed* as the code attribute of
1463 the top level <lu:Status> element, and the code attribute of the second level <lu:Status> element MUST be
1464 set with the following status code:
 - 1465 • ObjectIsEntity

1466 • The PS Provider MUST, if unable to find the targetted `Object`, respond with *Failed* as the code attribute of the
1467 top level `<lu:Status>` element, and the code attribute of the second level `<lu:Status>` element MUST be set
1468 with the following status code:

1469 • `CannotFindObject`

1470 • When the targeted `Object` has no child `Objects`, the PS provider MUST respond with *OK* as the code attribute of
1471 the top level `<lu:Status>` element with no `<Object>` element in the response.

1472 3.17. Retrieving Info

1473 A WSC uses the `<GetObjectInfoRequest>` message to retrieve information for a particular `Object`. An `Object`'s
1474 information includes all the child elements and attributes of the `<Object>` element, except for either `<Object>` or
1475 `<ObjectRef>` elements.

1476 Note that if a WSC wants to get child members's `Objects` of the particular `Object`, the WSC MUST use
1477 `<ListMembersRequest>` message (see [Section 3.16](#)).

1478 3.17.1. `wsa:Action` Values

1479 `<GetObjectInfoRequest>` messages MUST include a `<wsa:Action>` SOAP header with the value of
1480 "urn:liberty:ps:2006-08:GetObjectInfoRequest". `<GetObjectInfoResponse>` messages MUST include a
1481 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:GetObjectInfoResponse".

1482 3.17.2. `GetObjectInfoRequest` Message

1483 The WSC indicates to the PS provider the `Object` of interest by specifying that `<Object>` element's `ObjectID` in
1484 the `<TargetObjectID>` element.

1485 The `<GetObjectInfoRequest>` message has the complex type `GetObjectInfoRequestType`, which extends
1486 `RequestAbstractType` and adds the following element:

1487 `<TargetObjectID>` **[Required]** The `<TargetObjectID>` element is used to convey an `ObjectID` of the target
1488 `Object` of which information is requested.

1489 `<Subscription>` **[Optional]** The `<Subscription>` element is used to indicate to the PS provider that the
1490 WSC desires that the PS provider send a notification if and when the information (but not
1491 membership) of the targeted `Object` changes.

1492 The schema declaration for the <GetObjectInfoRequest> message is shown below.

```

1493
1494 <!-- Declaration of GetObjectInfoRequest element -->
1495 <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
1496 <!-- Definition of GetObjectInfoRequestType -->
1497 <xs:complexType name="GetObjectInfoRequestType">
1498   <xs:complexContent>
1499     <xs:extension base="RequestAbstractType">
1500       <xs:sequence>
1501         <xs:element ref="TargetObjectID"/>
1502         <xs:element ref="Subscription" minOccurs="0"/>
1503       </xs:sequence>
1504     </xs:extension>
1505   </xs:complexContent>
1506 </xs:complexType>
1507
```

1508 The following is an example of a <GetObjectInfoRequest> message.

```

1509
1510 <GetObjectInfoRequest>
1511   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1512 </GetObjectInfoRequest>
1513
```

1514 3.17.3. GetObjectInfoResponse Message

1515 A PS provider responds to a <GetObjectInfoRequest> message with a <GetObjectInfoResponse> message,
1516 in which the specified Object's information is conveyed.

1517 The <GetObjectInfoResponse> message has the complex type **GetObjectInfoResponseType**, which extends
1518 **ResponseAbstractType** and adds the following elements:

1519 <Object> **[Optional]** The <Object> element is used to convey the information of the requested Object.

1520 The schema declaration for the <GetObjectInfoResponse> message is shown below.

```

1521
1522 <!-- Declaration of GetObjectInfoResponse element -->
1523 <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
1524 <!-- Definition of GetObjectInfoResponseType -->
1525 <xs:complexType name="GetObjectInfoResponseType">
1526   <xs:complexContent>
1527     <xs:extension base="ResponseAbstractType">
1528       <xs:sequence>
1529         <xs:element ref="Object" minOccurs="0"/>
1530       </xs:sequence>
1531     </xs:extension>
1532   </xs:complexContent>
1533 </xs:complexType>
1534
```

1535 The following is an example of a <GetObjectInfoResponse> to the <GetObjectInfoRequest> message above.

```

1536
1537 <GetObjectInfoResponse>
1538   <Status code="OK"/>
1539   <Object NodeType="urn:liberty:ps:collection">
1540     <ObjectID>https://ps.com/eiruvoie</ObjectID>
1541     <DisplayName>Soccer Team</DisplayName>
1542   </Object>
1543 </GetObjectInfoResponse>
1544
```

1545 **3.17.4. Processing Rules**

1546 A PS provider:

- 1547 • MUST return the <Object> corresponding to the <TargetObjectID> element on the <GetObjectInfoRequest>
1548 message.
- 1549 • MUST, if it can not find the targetted Object, respond with *Failed* as the code attribute of the top level
1550 <lu:Status> element, and the code attribute of the second level <lu:Status> element MUST be set with
1551 the following status code:
 - 1552 • CannotFindObject
- 1553 • MUST NOT respond with any child <Object> and/or <ObjectRef> elements of the specified <Object>.

1554 **3.18. Updating Info**

1555 A WSC may wish to update or modify the information for an Object, e.g. to change a DisplayName etc.

1556 A WSC uses the <SetObjectInfoRequest> message to update the information for a particular Object. Updateable
1557 information does not include <Object> element, <ObjectRef> element, NodeType attribute, CreatedDateTime
1558 attribute, and ModifiedDateTime attribute.

1559 Note that if a WSC wants to insert a child Object to the particular Object, the WSC MUST use
1560 <AddToCollectionRequest> message (see [Section 3.14](#)). Also note that if a WSC wants to remove a child
1561 Object from the particular Object, the WSC MUST use <RemoveFromCollectionRequest> message (see
1562 [Section 3.15](#)).

1563 **3.18.1. wsa:Action Values**

1564 <SetObjectInfoRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1565 "urn:liberty:ps:2006-08:SetObjectInfoRequest". <SetObjectInfoResponse> messages MUST include a
1566 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:SetObjectInfoResponse".

1567 **3.18.2. SetObjectInfoRequest Message**

1568 The WSC specifies <Object> elements of the target Object for updating. These <Object> elements MUST
1569 NOT have child <Object> and/or <ObjectRef> elements. Also, these <Object> elements MUST NOT have
1570 CreatedDateTime and ModifiedDateTime attributes.

1571 The <SetObjectInfoRequest> message has the complex type **SetObjectInfoRequestType**, which extends
1572 **RequestAbstractType** and adds the following element:

1573 <Object> **[Required]** The <Object> element is used to convey the updated data of the Object to be updated.

1574 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
1575 desires that the PS provider send a notification if and when the information of the targeted
1576 Object changes.

1577 The schema declaration for the <SetObjectInfoRequest> message is shown below.

```

1578
1579 <!-- Declaration of SetObjectInfoRequest element -->
1580 <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
1581 <!-- Definition of SetObjectInfoRequestType -->
1582 <xs:complexType name="SetObjectInfoRequestType">
1583   <xs:complexContent>
1584     <xs:extension base="RequestAbstractType">
1585       <xs:sequence>
1586         <xs:element ref="Object" maxOccurs="unbounded"/>
1587         <xs:element ref="Subscription" minOccurs="0"/>
1588       </xs:sequence>
1589     </xs:extension>
1590   </xs:complexContent>
1591 </xs:complexType>
1592

```

1593 The following is an example of a <SetObjectInfoRequest> message in which the WSC is changing the value of
1594 the <DisplayName> element for the specified <Object>. The previously existing value for this element would be
1595 overwritten.

```

1596
1597 <SetObjectInfoRequest>
1598   <Object NodeType="urn:liberty:ps:collection">
1599     <ObjectID>https://ps.com/eiruvoie</ObjectID>
1600     <DisplayName>Baseball Team</DisplayName>
1601   </Object>
1602 </SetObjectInfoRequest>
1603

```

1604 3.18.3. SetObjectInfoResponse Message

1605 A PS provider responds to a <SetObjectInfoRequest> message with a <SetObjectInfoResponse> message.

1606 The <SetObjectInfoResponse> message has the type of **ResponseAbstractType**

1607 The schema declaration for the <SetObjectInfoResponse> message is shown below.

```

1608
1609 <!-- Declaration of SetObjectInfoResponse element -->
1610 <xs:element name="SetObjectInfoResponse" type="ResponseAbstractType"/>
1611

```

1612 The following is an example of a <SetObjectInfoResponse> to the <SetObjectInfoRequest> message above.

```

1613
1614 <SetObjectInfoResponse>
1615   <Status code="OK"/>
1616 </SetObjectInfoResponse>
1617

```

1618 3.18.4. Processing Rules

1619 A PS provider:

- 1620 • MUST, if it can not find the target Object specified with the ObjectID, respond with *Failed* as the code attribute
- 1621 of the top level <lu:Status> element, and the code attribute of the second level <lu:Status> element MUST
- 1622 be set with the following status code:

- 1623 • CannotFindObject

1624 • MUST, if it finds that the value of the specified `NodeType` attribute is different from the value of the `NodeType`
1625 attribute of the target `Object` specified with the `ObjectID`, respond with *Failed* as the code attribute of the top
1626 level `<lu:Status>` element, and the code attribute of the second level `<lu:Status>` element MUST be set with
1627 the following status code:

1628 • `InvalidNodeType`

1629 • MUST, if it finds that a WSC specifies the `<Object>` element, `<ObjectRef>` element, `CreatedDateTime`
1630 attribute, or `ModifiedDateTime`, ignore these elements and/or attributes.

1631 3.19. Querying Objects

1632 A WSC may wish to have returned to it a list of `Objects` that meet some criteria. The `<QueryObjectsRequest>`
1633 message allows the WSC to indicate this of the PS provider. The criteria to be met are specified in the `<Filter>`
1634 element in the `<QueryObjectsRequest>` element.

1635 Note: The `<ListMembersRequest>` message can be considered a specialization of the `<QueryObjectsRequest>`
1636 message, in which the criteria to be met is that the returned `Objects` are members of the specified group `Object`.
1637 The `<QueryObjectsRequest>` message allows a WSC to pose more generalized queries, e.g. to which groups does
1638 a specific user belong?

1639 3.19.1. wsa:Action Values

1640 `<QueryObjectsRequest>` messages MUST include a `<wsa:Action>` SOAP header with the value of
1641 "urn:liberty:ps:2006-08:QueryObjectsRequest". `<QueryObjectsResponse>` messages MUST include a
1642 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:QueryObjectsResponse".

1643 3.19.2. QueryObjectsRequest Message

1644 The WSC specifies criteria of its interest in the `<Filter>` element.

1645 The `<QueryObjectsRequest>` message has the complex type `QueryObjectsRequestType`, which extends
1646 `RequestAbstractType` and adds the following elements:

1647 `<Filter>` **[Required]** The `<Filter>` element is used to convey criteria for matching `Object` elements that a WSC
1648 is interested in.

1649 `<Subscription>` **[Optional]** The `<Subscription>` element is used to indicate to the PS provider that the WSC
1650 desires that the PS provider send a notification if and when the set of objects that satisfy the
1651 specified filter changes.

1652 `Count` **[Optional]** The `Count` attribute specifies how many `<Object>` elements should be returned in a
1653 `<QueryObjectsResponse>` message. See [Section 3.19.2.2](#) for more detail.

1654 `Offset` **[Optional]** The `Offset` attribute specifies from which element to continue, when requesting for more
1655 data. See [Section 3.19.2.2](#) for more detail.

1656 The schema declaration for the `<QueryObjectsRequest>` message is shown below.

```

1657
1658 <!-- Declaration of QueryObjectsRequest element -->
1659 <xs:element name="QueryObjectsRequest" type="QueryObjectsRequestType" />
1660 <!-- Definition of QueryObjectsRequestType -->
1661 <xs:complexType name="QueryObjectsRequestType">
1662   <xs:complexContent>
1663     <xs:extension base="RequestAbstractType">
1664       <xs:sequence>
1665         <xs:element name="Filter" type="xs:string" />
1666         <xs:element ref="Subscription" minOccurs="0" />
1667       </xs:sequence>
1668       <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional" />
1669       <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional" />
1670     </xs:extension>
1671   </xs:complexContent>
1672 </xs:complexType>
1673
  
```

1674 3.19.2.1. Filter Element

1675 The value of the `<Filter>` element SHOULD be specified with an XPath expression.

1676 For instance, if the WSC wants to get all the `<Object>` elements with a `NodeType` attribute with a value of
 1677 `urn:liberty:ps:collection` (i.e., the WSC wants to get all the group `Objects`), the WSC can specify the value of
 1678 the `<Filter>` element as `"//Object[@NodeType='urn:liberty:ps:collection']"`.

1679 The following is an example of a `<QueryObjectsRequest>` message with which the WSC requests to get all the
 1680 `<Object>` elements that have `urn:liberty:ps:entity` as the value of `NodeType` element.

```

1681
1682 <QueryObjectsRequest>
1683   <Filter>//Object[@NodeType='urn:liberty:ps:entity']</Filter>
1684 </QueryObjectsRequest>
1685
  
```

1686 3.19.2.2. Count and Offset Attributes

1687 When the result of specified criteria has multiple `Objects`, the WSC may desire to be sent their `<Object>` elements in
 1688 smaller sets (i.e., a smaller number of elements at a time). This is achieved by using the `Count` and `Offset` attributes
 1689 of the `<QueryObjectsRequest>` element.

1690 The `Count` attribute defines how many `<Object>` elements should be returned in a `<QueryObjectsResponse>`
 1691 message.

1692 The `Offset` attribute specifies from which element to continue, when requesting for more data. The default value is
 1693 zero, which refers to the first `<Object>` element.

1694 3.19.3. QueryObjectsResponse Message

1695 A PS provider responds to a `<QueryObjectsRequest>` message with a `<QueryObjectsResponse>` message.
 1696 The `<QueryObjectsResponse>` contains the `<Object>` elements that meet the criteria specified in the `<Filter>`
 1697 element of the `<QueryObjectsRequest>` message.

1698 The `<QueryObjectsResponse>` message has the complex type `QueryObjectsResponseType`, which extends
 1699 `ResponseAbstractType` and adds the following element:

1700 <Object> **[Optional]** The <Object> element is used to convey data of zero or more Objects that the requested
 1701 criteria are met to.

1702 The schema declaration for the <QueryObjectsResponse> message is shown below.

```
1703
1704 <!-- Declaration of QueryObjectsResponse element -->
1705 <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
1706 <!-- Definition of QueryObjectsResponseType -->
1707 <xs:complexType name="QueryObjectsResponseType">
1708   <xs:complexContent>
1709     <xs:extension base="ResponseAbstractType">
1710       <xs:sequence>
1711         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
1712       </xs:sequence>
1713     </xs:extension>
1714   </xs:complexContent>
1715 </xs:complexType>
1716
1717
```

1718 The following is an example of a <QueryObjectsResponse> to the <QueryObjectsRequest> message above.

```
1719
1720 <QueryObjectsResponse>
1721   <Status code="OK"/>
1722   <Object NodeType="urn:liberty:ps:entity">
1723     <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1724     <DisplayName>Mary</DisplayName>
1725   </Object>
1726   <Object NodeType="urn:liberty:ps:entity">
1727     <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1728     <DisplayName>Bob</DisplayName>
1729   </Object>
1730   <Object NodeType="urn:liberty:ps:entity">
1731     <ObjectID>https://ps.com/reremvls</ObjectID>
1732     <DisplayName>Nick</DisplayName>
1733   </Object>
1734   <Object NodeType="urn:liberty:ps:entity">
1735     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1736     <DisplayName>JoJo</DisplayName>
1737   </Object>
1738   <Object NodeType="urn:liberty:ps:entity">
1739     <ObjectID>https://ps.com/sdosafms</ObjectID>
1740     <DisplayName>Taro</DisplayName>
1741   </Object>
1742   <Object NodeType="urn:liberty:ps:entity">
1743     <ObjectID>https://ps.com/lgsdfsf</ObjectID>
1744     <DisplayName>Hanako</DisplayName>
1745   </Object>
1746 </QueryObjectsResponse>
1747
```

1748 3.19.4. Processing Rules

- 1749 • If a WSC specifies the value of the <Filter> element through an XPath expression, the value SHOULD begin
 1750 with the expression "//Object".
- 1751 • All the <Object> elements that are responded in the <QueryObjectsResponse> message from a PS provider
 1752 MUST NOT contain any child <Object> and/or <ObjectRef> elements.

1753 • If a PS provider cannot find any Objects that meets the criteria that a WSC specifies in the request, the PS
1754 provider MUST respond *OK* as the code attribute of the top level <lu:Status> element, and the code attribute
1755 of the second level <lu:Status> element MUST be set with the following status code:

1756 • NoResults

1757 3.20. Testing Membership

1758 A WSC may wish to pose a question of the Object tree structure and have the results of that question returned rather
1759 than the Object tree itself (as the <ListMembersRequest> or <QueryObjectsRequest> messages support). For
1760 instance, the WSC might wish to ask whether a specified individual (or more generally any Object) is a member of
1761 a particular specified group Object. This scenario will be common when the owning user has defined some access
1762 control policy in terms of membership in some group (e.g. 'allow members of my soccer team to view these photos').
1763 If and when some other user tries to access the resource in question, the WSC will need to determine if they are entitled
1764 (e.g. whether or not they are on the soccer team).

1765 The <TestMembershipRequest> allows a WSC to pose the question 'Is user X a member of group Y?'

1766 3.20.1. wsa:Action Values

1767 <TestMembershipRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1768 "urn:liberty:ps:2006-08:TestMembershipRequest". <TestMembershipResponse> messages MUST include a
1769 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:TestMembershipResponse".

1770 3.20.2. TestMembershipRequest Message

1771 The target parent group Object for which the membership of another Object is being tested is indicated by the
1772 value of the <TargetObjectID> element within the <TestMembershipRequest> message. The Object for which
1773 membership is being tested is specified by the <Token> element within the <TestMembershipRequest> message.

1774 The <TestMembershipRequest> message has the complex type **TestMembershipRequestType**, which extends
1775 **RequestAbstractType** and adds the following elements:

1776 <TargetObjectID> **[Optional]** The <TargetObjectID> element is used to convey the ObjectID of the target
1777 group Object for which the membership of a user is being tested.

1778 <Token> **[Required]** The <Token> element is used to convey an identity token of a user for which membership is
1779 being tested.

1780 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
1781 desires that the PS provider send a notification if and when results of the test changes.

1782 The schema declaration for the <TestMembershipRequest> message is shown below.

```

1783
1784 <!-- Declaration of TestMembershipRequest element -->
1785 <xs:element name="TestMembershipRequest" type="TestMembershipRequestType"/>
1786 <!-- Definition of TestMembershipRequestType -->
1787 <xs:complexType name="TestMembershipRequestType">
1788   <xs:complexContent>
1789     <xs:extension base="RequestAbstractType">
1790       <xs:sequence>
1791         <xs:element ref="TargetObjectID" minOccurs="0"/>
1792         <xs:element ref="Token"/>
1793         <xs:element ref="Subscription" minOccurs="0"/>
1794       </xs:sequence>
1795     </xs:extension>
1796   </xs:complexContent>
1797 </xs:complexType>
    
```

1798 The following is an example of a <TestMembershipRequest> message.

```

1799
1800 <TestMembershipRequest>
1801   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1802   <Token>
1803
1804   </Token>
1805 </TestMembershipRequest>
    
```

1806 3.20.3. TestMembershipResponse Message

1807 The PS returns the result of the specified membership test in a <Result> element within a
 1808 <TestMembershipResponse> message.

1809 The <TestMembershipResponse> message has the complex type **TestMembershipResponseType**, which ex-
 1810 tends **ResponseAbstractType** and adds the following elements:

1811 <Result> **[Required]** The <Result> element is used to convey the result of the specified membership test. This
 1812 element has a type of **ResultType**, which is derived from **xs:boolean**.

1813 The schema declarations for the <TestMembershipResponse> message and the <Result> element are shown
 1814 below.

```

1815
1816 <!-- Definition of ResultType -->
1817 <xs:complexType name="ResultType">
1818   <xs:complexContent>
1819     <xs:extension base="xs:boolean"/>
1820   </xs:complexContent>
1821 </xs:complexType>
1822
1823 <!-- Declaration of TestMembershipResponse element -->
1824 <xs:element name="TestMembershipResponse" type="TestResponseType"/>
1825 <!-- Definition of TestMembershipResponseType -->
1826 <xs:complexType name="TestMembershipResponseType">
1827   <xs:complexContent>
1828     <xs:extension base="ResponseAbstractType">
1829       <xs:sequence>
1830         <xs:element name="Result" type="ResultType" minOccurs="0"/>
1831       </xs:sequence>
1832     </xs:extension>
1833   </xs:complexContent>
1834 </xs:complexType>
    
```

1835 The following is an example of a `<TestMembershipResponse>` message.

```
1836
1837     <TestMembershipResponse>
1838         <Status code="OK"/>
1839         <Result>true</Result>
1840     </TestMembershipResponse>
```

1841 **3.20.4. Processing Rules**

1842 If the `<TargetObjectID>` element specifies the `ObjectID` of an `Object` with a `NodeType` attribute of
1843 "urn:liberty:ps:entity", the PS provider **MUST** respond with *Failed* as the code attribute of the top level `<lu:Status>`
1844 element, and the code attribute of the second level `<lu:Status>` element **MUST** be set with the following status
1845 code:

1846 • `ObjectIsEntity`

1847 **3.21. Resolving Objects**

1848 Once a WSC has an `ObjectID` for an entity, it will often desire to communicate with other providers about that user.
1849 To do so, the WSC will first need an identity token for that user. The process of converting an object identifier into an
1850 identity token is referred to as *resolving* the identity token.

1851 **3.21.1. wsa:Action Values**

1852 `<ResolveIdentifierRequest>` messages **MUST** include a `<wsa:Action>` SOAP header with the value of
1853 "urn:liberty:ps:2006-08:ResolveIdentifierRequest". `<ResolveIdentifierResponse>` messages **MUST** include a
1854 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:ResolveIdentifierResponse".

1855 **3.21.2. ResolveIdentifierRequest Message**

1856 The WSC can use the `<ResolveIdentifierRequest>` message to ask the PS provider to resolve the specified
1857 `ObjectID` in the `<TargetObjectID>` element, into an appropriate identity token.

1858 An `<ResolveIdentifierRequest>` message consists of one or more `<ResolveInput>` elements. If multiple
1859 `<ResolveInput>` elements are included in a request, then each **MUST** contain a `reqID` attribute so that the response
1860 contents can be correlated to them.

1861 The WSC **MAY** specify its requirements of the identity token through the `<sec:TokenPolicy>`

1862 The `<ResolveIdentifierRequest>` message has the complex type **ResolveIdentifierRequestType**, which
1863 extends **RequestAbstractType** and adds the following elements:

1864 `<ResolveInput>` [**One or more**] The object for which the WSC desires an identity token be resolved.

1865 The schema declaration for the <ResolveIdentifierRequest> message is shown below.

```

1866
1867 <!-- Declaration of ResolveIdentifierRequest element -->
1868 <xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
1869 <!-- Definition of ResolveIdentifierRequestType -->
1870 <xs:complexType name="ResolveIdentifierRequestType">
1871   <xs:complexContent>
1872     <xs:extension base="RequestAbstractType">
1873       <xs:sequence>
1874         <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
1875       </xs:sequence>
1876     </xs:extension>
1877   </xs:complexContent>
1878 </xs:complexType>
1879

```

1880 3.21.2.1. ResolveInput element

1881 The <ResolveInput> element is of complex type <ResolveInputType>, which extends the complex type
1882 <MappingInputType> defined in [[LibertyAuthn](#)] to add the following element:

1883 <TargetObjectID> **[Required]** The <TargetObjectID> conveys the ObjectID of the target entity Object for
1884 which the WSC is requesting an identity token be resolved.

1885 The schema declaration for the <ResolveInput> element is shown below.

```

1886
1887 <!-- Declaration of ResolveInput element -->
1888 <xs:element name="ResolveInput" type="ResolveInputType"/>
1889 <!-- Definition of ResolveInputType -->
1890 <xs:complexType name="ResolveInputType">
1891   <xs:complexContent>
1892     <xs:extension base="ims:MappingInputType">
1893       <xs:sequence>
1894         <xs:element ref="TargetObjectID"/>
1895       </xs:sequence>
1896     </xs:extension>
1897   </xs:complexContent>
1898 </xs:complexType>
1899

```

1900 The semantics and processing rules of the elements and attributes are as defined in [[LibertyAuthn](#)].

1901 The following is an example of a <ResolveIdentifierRequest> message in which the WSC is requesting that an
1902 identity token for the Object identified by the specified ObjectID be returned. The WSC is indicating that it desires
1903 a transient identifier for the identity token.

```

1904
1905 <ResolveIdentifierRequest>
1906   <ResolveInput>
1907     <TargetObjectID>https://ps.com/lgsd fsfd</TargetObjectID>
1908     <sec:TokenPolicy>
1909       <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
1910     </sec:TokenPolicy>
1911   </ResolveInput>
1912 </ResolveIdentifierRequest>
1913

```

1914 3.21.3. ResolveIdentifierResponse Message

1915 A PS provider responds to a <ResolveIdentifierRequest> message with a <ResolveIdentifierResponse>
 1916 message. The PS provider returns the identity tokens corresponding to the Objects specified by the ObjectID in the
 1917 TargetObjectID element on the <ResolveIdentifierRequest> message.

1918 The schema declaration for the <ResolveIdentifierResponse> message is shown below.

```

1919
1920 <!-- Declaration of ResolveIdentifierResponse element -->
1921 <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType" />
1922 <!-- Definition of ResolveIdentifierResponseType -->
1923 <xs:complexType name="ResolveIdentifierResponseType">
1924   <xs:complexContent>
1925     <xs:extension base="ResponseAbstractType">
1926       <xs:sequence>
1927         <xs:element ref="ResolveOutput" maxOccurs="unbounded" />
1928       </xs:sequence>
1929     </xs:extension>
1930   </xs:complexContent>
1931 </xs:complexType>
1932
```

1933 3.21.3.1. ResolveOutput element

1934 Each <ResolveOutput> element consists of an identity token (in the form of a <sec:Token>).

1935 The element is of complex type MappingOutputType defined in [[LibertyAuthn](#)]. Returned tokens are correlated with
 1936 input object identifiers through the reference mechanism defined in [[LibertyAuthn](#)]

1937 The declaration for the <ResolveOutput> element is shown below.

```

1938
1939 <!-- Declaration of ResolveOutput element -->
1940 <xs:element name="ResolveOutput" type="ims:MappingOutputType" />
1941
```

1942 The following is an example of a <ResolveIdentifierResponse> to the <ResolveIdentifierRequest>
 1943 message above.

```

1944
1945 <ResolveIdentifierResponse>
1946   <Status code="OK" />
1947   <ResolveOutput>
1948     <Token>
1949
1950   </Token>
1951 </ResolveOutput>
1952 </ResolveIdentifierResponse>
```

1953 3.21.4. Processing Rules

1954 • Upon receiving a <ResolveIdentifierRequest> from a SP carrying <TargetObjectID> elements that
 1955 correspond to existing Object, the PS provider MUST endeavor to return to the SP an appropriate identity token
 1956 corresponding to that Object.

1957 To do so, the PS provider MAY itself send a <ims:IdentityMappingRequest> message to the relevant identity
 1958 provider for the Object in question, specifying the requesting WSC as the target namespace.

- 1959 • If a PS provider can not resolve any of input objects into identity tokens, the PS provider MUST respond Failed
1960 as the code attribute of the top level <lu:Status> element, and MAY use second level <lu:Status> elements
1961 that MUST contain a ref attribute equal to the associated <ResolveInput>'s reqID attribute. The second level
1962 <lu:Status> elements corresponding to failed inputs MUST be set with the following status code:
- 1963 • CannotResolveToken
- 1964 • If a PS provider is unable to resolve all input objects into identity tokens, the PS provider MUST respond
1965 PartialSuccess as the code attribute of the top level <lu:Status> element, and SHOULD use second level
1966 <lu:Status> elements that MUST contain a ref attribute equal to the associated <ResolveInput>'s reqID
1967 attribute. The second level <lu:Status> elements corresponding to failed inputs MUST be set with the following
1968 status code:
- 1969 • CannotResolveToken

1970 **4. Interaction with Users**

1971 Before a user can be added to another user's PS resource, appropriate federations may need to be established between
1972 various providers. Before this can happen, it will typically be necessary for the user to *visit* these providers in order to
1973 kick-off the process. The mechanism by which this prompt occurs is referred to here as an *invitation*.

1974 Except for special cases where a user can be added to a user's PS resource without the participation of that user (as
1975 would be possible if the PS resource owning user happened to know an identifier for the other user at some IDP), such
1976 an invitation is necessary. Supporting such user interactions is a key aspect of the PS role.

1977 **4.1. Model (Informative)**

1978 The invitation model is as follows:

1979 1. A user (visiting one of their SP's or their PS provider), decides that they wish to add some friend/contact/family
1980 member to their PS resource (likely in the context of enabling some specific interaction with that friend)

1981 2. An invitation (consisting of some human readable descriptive text explaining the context as well as some
1982 mechanism by which interaction can be kicked off) is created by the provider on behalf of the PS resource
1983 owning user (the *inviting* user).

1984 3. The invitation is delivered to the relevant user (the *invited* user).

1985 4. The invited user examines the invitation and decides whether or not they wish to accept. If no, they take no
1986 further steps. If so, they proceed with the indicated mechanism to interact with the relevant providers (being
1987 given appropriate information and consent mechanisms at each step).

1988 5. The invited user is added to the inviting user's PS resource.

1989 6. The invited user can be directed to the SP to access the resource in question.

1990 It is also possible for a user to add their friends to their PS list through whatever browser-interface that PS provider
1991 makes available. In essence, these friends would be added independent of any particular interaction context, but
1992 available for future interactions once added. In such a case there will still need to be an invitation sent to the friend
1993 being added (with the same proviso for a known identifier) in order to facilitate the establishment of a federation
1994 between the PS and the invited user's IDP, but the process can be simpler because there is no initiating SP.

1995 **4.2. Additional Federations for Sharing of Identity Services**

1996 When the invitation process is initiated from an SP and the resource being shared is browser-based, the normal result
1997 of successful interaction is that federated identifiers for the invited user are established between their IDP and both the
1998 PS provider and the initiating SP.

1999 When, rather than browser-based, the resource being shared is an identity service (e.g. a user's online presence) it
2000 may be necessary for an additional federation for the invited user be established between their IDP and the Discovery
2001 Service (DS) (see [[LibertyDisco](#)]) of the inviting user.

2002 Ultimately, when the shared resource is some identity service hosted by a WSP on behalf of one user (who has decided
2003 to make access to it available), it will be accessed by some WSC on behalf of the other user (that to which access
2004 privileges have been granted).

2005 To facilitate this operation, the DS of the inviting user will likely need to provide an appropriate WSP
2006 <EndpointReference> to the WSC upon that WSC querying for relevant (associated with the inviting user
2007 and of a particular service type) WSPs. If the inviting user's policy is such that merely the fact of existence or
2008 location of their identity data (beyond its actual value) warrants protection, then the DS should apply access control

2009 mechanisms to such WSC queries. To do so, it may need to have an identifier for the requesting user so that an
2010 appropriate access control decision can be made.

2011 In this sense, the existence and location of a user's identity services (as exemplified in the endpoint references pointing
2012 there), are identity resources like any other identity service and may require appropriate federations be established to
2013 enable access control.

2014 If policy is such that no such fine-grained access control for <EndpointReference>s is deemed necessary, then
2015 establishing a federation between the DS of the inviting user and the IDP of the invited user will not be necessary.

2016 **4.3. Consent Model**

2017 A number of consent models governing the various federations established through the invitation process are possible.
2018 A few examples, amongst others, are listed below:

2019 One possibility would be for an invited user, when federating their IDP to the PS provider of a friend upon an invitation
2020 from that friend, to specify that the establishment of subsequent federations between service providers and their IDP
2021 need not require additional consent. Such blanket consent, while optimizing usability, could present unforeseen risks,
2022 e.g. giving a user the permission, even if never taken advantage of, to view offensive online material.

2023 At the other extreme, a user could request that they be asked for specific consent for each and every such federation.

2024 Additionally, a user could specify that their IDP be allowed to establish "provisional" federated identifiers with other
2025 SPs, but that they expect to be given the opportunity to give explicit consent to these federations if and when they
2026 attempt to use them. The identifiers are provisional in the sense that the IDP will not actually agree to use them with
2027 the other provider until such consent is obtained, at which point they would become "real". This model would ensure
2028 that the user would not be presented with numerous requests for consent - consent would be obtained only when
2029 necessary.

2030 **4.4. Elements Supporting Invitation**

2031 The following elements support the invitation model. These elements are optional within the appropriate protocol
2032 messages.

2033 **4.4.1. PStoSPRedirectURL element**

2034 The SP MAY use the <PStoSPRedirectURL> element on <AddEntityRequest> and <AddKnownEntityRequest>
2035 messages to specify to the PS provider the URL (at the SP) to which that SP desires the invited user's user agent be
2036 directed after successful interaction and IDP federation has occurred. If and when the invited user's user agent has
2037 been sent to this URL, the SP MAY provide the invited user the designated access to the resource in question.

2038 The value of the <PStoSPRedirectURL> element MUST be such that, if and when a user agent is sent to this address
2039 from the PS provider, the SP can unambiguously determine the invitation to which the URL corresponds. The URL
2040 MUST be unique for each combination of the inviting user, the PS, and the invited user.

2041 **4.4.2. <SptoPSRedirectURL> element**

2042 The PS provider MAY use the <SptoPSRedirectURL> element on <AddEntityResponse> and
2043 <AddKnownEntityResponse> messages to specify to the SP the URL (at the PS provider) to which that PS
2044 desires the invited user's user agent be directed after successful initial interaction at the SP has occurred. If and when
2045 the invited user's user agent is sent to this URL, the PS provider will endeavour to establish a federation for that
2046 principal with the appropriate IDP.

2047 The value of the <SptoPSRedirectURL> element MUST be such that, if and when a user agent is sent to this address
2048 from the SP, the PS can unambiguously determine the invitation to which the URL corresponds. The URL MUST be
2049 unique for each combination of the inviting user (that owns the PS resource), the requesting SP, and the invited user.

2050 The PS provider MUST be prepared for the invited user to, at some point in the future, visit the URL provided in
2051 any specified <SPtoPSRedirectURL> element. As it may be some time before the invited user does respond, the PS
2052 provider SHOULD store this url for a reasonable length of time.

2053 **4.4.2.1. Processing Rules**

2054 If and when the invited user responds to the invitation, the SP:

2055 • MUST, after appropriately informing and 'consenting' the invited user, direct the user agent to the address
2056 previously specified within the <SPtoPSRedirectURL> element.

2057 Once the invited user has been redirected to the PS provider, the PS provider:

2058 • MUST determine the invited user's IDP (or preferred IDP if the invited user has multiple).

2059 MUST endeavor to establish a federated identity for that user with that IDP.

2060 • MAY obtain an identity token from the IDP for the invited user targeted for itself.

2061 • MUST redirect the invited user's agent to the address previously specified by any <PStoSPRedirectURL>
2062 element in the original <AddEntityRequest> or <AddKnownEntityRequest> message.

2063 Once the PS has redirected the invited user to the <PStoSPRedirectURL> address, the SP MAY choose to send
2064 a <samlp:AuthnRequest> message to the IDP asking for a <saml:AuthnStatement> attesting to that user's
2065 authentication status there.

2066 In its corresponding <samlp:Response>, the IDP SHOULD use the same subject identifier for this
2067 <saml:Assertion> as previously delivered to the SP within the identity token through the PS provider and
2068 the <Notify> message (unless any SP policy on the <samlp:AuthnRequest> message precludes this).

2069 **4.4.3. <QueryString> element**

2070 The <QueryString> element enables an alternative model for invited user interaction which is expected to better
2071 defend against identity theft attacks in which a valid email is spoofed to fool users into clicking an embedded
2072 URL. The invitation received by the invited user will contain a string carrying a SAML artifact (and potentially
2073 relay state info) representing a SAML <samlp:AuthnRequest> message created by the PS provider. The invited
2074 user can, if they choose, present this artifact string to their identity provider - which can then use the SAML
2075 <samlp:ArtifactResolve> message to retrieve the original <samlp:AuthnRequest> message from the PS
2076 provider.

2077 As the invited user visits their IDP by explicitly providing the address or using an existing bookmark, they can be more
2078 confident that the site is not spoofed. Once they are at their IDP and after presenting the SAML artifact, appropriate
2079 federations can be established for the invited user with the originating PS provider and SP.

2080 **4.4.3.1. Schema**

```
2081 <xs:element name="QueryString" type="QueryStringType"/>
2082   <xs:complexType name="QueryStringType">
2083     <xs:simpleContent>
2084       <xs:extension base="xs:string"/>
2085     </xs:simpleContent>
2086   </xs:complexType>
2087
```

2088 4.4.3.2. Formatting Rules

2089 The contents of the <QueryString> element MUST satisfy the formatting requirements of the URL encoding of the
2090 SAML Artifact Binding (see [[SAMLBind2](#)]) as identified by the URI:

2091 *urn:oasis:names:tc:SAML:2.0:artifact-04.*

2092 The following is an example of an <QueryString> element in which the PS Provider has included relay state
2093 information through an additional RelayState parameter.

```
2094  
2095 <QueryString>SAMLart=AAQAADWNEw5VT47wcO4zX%2FiEzMmFQv%3D&Re layState=0043bfc1bc45110dae170  
2096 04005b13a2b</QueryString>
```

2097 The contents of the above element would be communicated to the invited user by the SP (either directly or indirectly
2098 through the inviting user) for them to provide to their IDP.

2099 4.4.3.3. Processing Rules

2100 To support this invitation model, when responding to either a <AddEntityRequest> or <AddKnownEntityRequest>
2101 message, the PS provider:

- 2102 • MUST create an artifact string in accordance with the SAML Artifact Binding (see [[SAMLBind2](#)]).
- 2103 • MUST insert this string within an <QueryString> element in the <AddEntityResponse> or
2104 <AddKnownEntityResponse> message.
- 2105 • MUST be prepared for, at some point in the future, an IDP to send an <samlp:ArtifactResolve> message as a
2106 consequence of the invited user presenting the contents of any specified <QueryString> element to the IDP. As
2107 it may be some time before the invited user does present the artifact to their IDP, the PS provider SHOULD store
2108 the artifact for a reasonable length of time.
- 2109 MUST return the appropriate <samlp:AuthnRequest> message in its <samlp:ArtifactResponse> message.
- 2110 MAY, when the IDP returns a name identifier (either pre-existing or generated) for the invited user in its
2111 <samlp:Response> message, send an <ims:IdentityMappingRequest> message to the IDP requesting an
2112 identity token (targeted at itself) for the invited user unless it already has such a identity token.

2113 After receiving an <AddEntityResponse> or <AddKnownEntityResponse> message with an <QueryString>
2114 element, the SP:

- 2115 • MUST extract the contents of the <QueryString> element
- 2116 • MUST attempt to communicate the extracted string to the invited user.

2117 If and when the invited user presents the query string that it received from the SP to its IDP, the IDP:

- 2118 • MUST authenticate the user
- 2119 • MUST determine the identity of the PS provider from the artifact string *SourceID* and determine the addresses to
2120 which <samlp:ArtifactResolve> and <Response> messages are to be sent (e.g. through metadata).
- 2121 • MUST send an <samlp:ArtifactResolve> message to the PS provider and MUST process the retrieved
2122 <samlp:AuthnRequest> message in accordance with the SSO Profiles of SAML [[SAMLProf2](#)].

-
- 2123 • MUST create and deliver a `<samlp:Response>` message to the PS provider using a SAML front-channel binding
2124 (e.g. HTTP Redirect, HTTP POST, or HTTP Artifact).
- 2125 If the value of the presented query string included any relay state information, the binding by which the
2126 `<samlp:Response>` message is delivered to the PS MUST support the communication of this relay state
2127 information back to the PS provider.
- 2128 Once the invited user has been redirected to the PS provider and the PS provider has obtained the `<samlp:Response>`,
2129 the PS provider:
- 2130 • MUST extract the name identifier from within the `<Subject>` of the `<Assertion>`.
- 2131 • MUST determine the original `<AddEntityRequest>` or `<AddKnownEntityRequest>` message to which the
2132 returned identifier corresponds.
- 2133 • MUST use the appropriate federated name identifier for the user to obtain an identity token from the IDP for the
2134 invited user - this identity token targeted for itself.
- 2135 • MUST, unless the SP did not include a `<Subscription>` in its `<AddEntityRequest>` message, obtain an
2136 identity token from the IDP for the invited user targeted at the SP.
- 2137 MUST forward on the identity token just received from the IDP in a `<Notify>` message, specifying the
2138 `SubscriptionID` of the previous `<Subscription>` element.
- 2139 • MUST redirect the invited user's agent to the address previously specified by the `<PStoSPRedirectURL>` element
2140 in the original `<AddEntityRequest>` or `<AddKnownEntityRequest>` message.
- 2141 Once the invited user has been redirected to the `<PStoSPRedirectURL>` address, the SP:
- 2142 • MAY choose to send a `<samlp:AuthnRequest>` message to the IDP asking for a `<saml:AuthnStatement>`
2143 attesting to that user's authentication status there. In its `<Response>`, the IDP MUST use the same subject
2144 identifier for this `<saml:Assertion>` as previously delivered to the SP within the identity token through the
2145 PS provider and the `<Notify>` message.

2146 5. Sequence Examples

2147 Following are detailed sequence examples for:

- 2148 • setting access control against a PS group
- 2149 • checking group membership for access control
- 2150 • performing a collective operation against group members

2151 5.1. Policy definition

2152 The following sequences demonstrates examples of the messages exchanged when a user defines access control for
2153 some SP resource in terms of group membership

- 2154 1. Alice visits SPa and indicates that she wishes to allow a group of hers to view some resource there.
- 2155 2. SPa discovers Alice's PS Provider, PSa.
- 2156 3. SPa queries PSa for top-level Objects.

```
2157  
2158     <ListMembersRequest Structured="children" />  
2159
```

- 2160 4. PS responds with the Objects.

```
2161  
2162     <ListMembersResponse>  
2163     <Status code="OK" />  
2164     <Object NodeType="urn:liberty:ps:entity" >  
2165     <ObjectID="https://psa.com/sdfhgusfsf" />  
2166     <DisplayName>Bob</DisplayName>  
2167     </Object>  
2168     <Object NodeType="urn:liberty:ps:entity" >  
2169     <ObjectID="https://psa.com/itndojd" />  
2170     <DisplayName>Mary</DisplayName>  
2171     </Object>  
2172     <Object NodeType="urn:liberty:ps:collection" >  
2173     <ObjectID="https://psa.com/sijfsfsf" />  
2174     <DisplayName>Work Friends</DisplayName>  
2175     </Object>  
2176     <Object NodeType="urn:liberty:ps:collection" >  
2177     <ObjectID="https://psa.com/lsdjfojd" />  
2178     <DisplayName>Soccer Team</DisplayName>  
2179     </Object>  
2180     </ListMembersResponse>  
2181
```

- 2182 5. SPs displays the list to Alice.
- 2183 6. Alice specifies that members of the group called 'Work Friends' should be able to access the resource in question.
- 2184 7. SPa defines appropriate permissions against the 'Work Friends' group's ObjectID of 'https://psa.com/sijfsfsf'.
2185 If and when somebody tries to access Alice's resource in question, at that point SPa will need to determine if that
2186 individual is a member of the group Object with this ObjectID. See the following example in [Section 5.2](#) for
2187 the sequences of messages.

2188 Rather than defining permissions against the ObjectID, the service provider could have chosen to obtain identity
2189 tokens (using a sequence of <ListMembersRequest> and <ResolveIdentifierRequest> messages) for all
2190 current members of the 'Work Friends' group and then define access control rules directly against the relevant
2191 identifiers. This may not be appropriate if the membership of the group in question is expected to change.

2192 5.2. AccessControl

2193 The following is an example of the use-case in which an SP uses group membership information for controlling access
2194 to resources that it holds. In the use-case, Alice has defined access rules to some resources at SPa/WSCa based on
2195 membership in a group she maintains at PSa. Bob is a friend of Alice. When Bob appears at the SPa and tries to
2196 access the resource in question, the SPa must determine if Bob is a member of the group.

2197 1. Bob shows up at SPa and tries to access the resource in question.

2198 2. SPa asks 'Who are you?'.

2199 3. Bob says 'Ask IDPb'.

2200 4. SPa redirects Bob to IDPb with AuthnRequest.

```
2201 <samlp:AuthnRequest
2202 ID="NTT7630E00861279F0ADC63E241D0926D0B"
2203 Version="2.0" IssueInstant="...">
2204 <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2205 https://spa.com
2206 </saml:Issuer>
2207 <samlp:NameIDPolicy
2208 Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
2209 </samlp:AuthnRequest>
2210
2211
```

2212 5. IDPb authenticates Bob.

2213 6. IDPb sends a Response to SPa with an AuthnStatement carrying a name identifier for Bob.

```
2214 <samlp:Response
2215 ID="NTT3F633E3F712BAC4B0804714431D46D7B"
2216 InResponseTo="NTT7630E00861279F0ADC63E241D0926D0B"
2217 Version="2.0" IssueInstant="...">
2218 <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2219 https://idpb.com
2220 </saml:Issuer>
2221 <samlp:Status>
2222 <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
2223 </samlp:Status>
2224 <saml:Assertion
2225 Version="2.0" IssueInstant="..."
2226 ID="NTT02062BBDE3E97EF0749828BCB8C15DFB">
2227 <saml:Issuer
2228 Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2229 https://idpb.com
2230 </saml:Issuer>
2231 <saml:Subject>
2232 <saml:NameID
2233 NameQualifier="https://idpb.com"
2234 Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
2235 e0b735bf9d1f3959241d3584733d704c
2236 </saml:NameID>
2237 </saml:Subject>
2238 <saml:AuthnStatement
2239 AuthnInstant="..." SessionIndex="..."
2240 <saml:AuthnContext>AuthnContext goes here</saml:AuthnContext>
2241 </saml:AuthnStatement>
2242 </saml:Assertion>
2243 </samlp:Response>
2244
2245
```

2246 7. SPa sends IDPb an <ims:IdentityMappingRequest>, providing the previous name identifier for Bob and
2247 specifying PSa as the target namespace.

```
2248 <ims:IdentityMappingRequest>
2249 <ims:MappingInput>
2250 <sec:TokenPolicy SPNameQualifier="https://psa.com"/>
2251 <sec:Token>
2252 <saml:NameID NameQualifier="https://idpb.com"
2253 Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
2254 e0b735bf9d1f3959241d3584733d704c
2255 </saml:NameID>
2256 </sec:Token>
2257 </ims:MappingInput>
2258 </ims:IdentityMappingRequest>
2259
2260
```

2261 8. IDPb returns an appropriate mapped identifier for Bob that PSa will recognize.

```
2262 <ims:IdentityMappingResponse>
2263 <ims:MappingOutput>
2264 <sec:Token>
2265 <saml:Assertion>
2266 identity token for Bob in PSa's namespace
2267 </saml:Assertion>
2268 </sec:Token>
2269 </ims:MappingOutput>
2270 </ims:IdentityMappingResponse>
2271
2272
```

2273 9. As Alice has defined her access control rules in terms of a group maintained at PSa, SPa knows how to invoke
2274 PSa. SPa sends a query to PSa questioning Bob's membership in the group in question.

```
2275 <TestMembershipRequest>
2276 <TargetObjectID>https://ps.com/sijfsfs</TargetObjectID>
2277 <sec:Token>
2278 <saml:Assertion>
2279 identity token for Bob in PSa's namespace
2280 </saml:Assertion>
2281 </sec:Token>
2282 </TestMembershipRequest>
2283
2284
```

2285 10. PSa extracts the identity token, might decrypt the encrypted identifier in the identity token, looks up the specified
2286 group, and finds Bob's entry.

2287 11. PSa returns 'true' to SPa.

```
2288 <TestMembershipResponse>
2289 <Status code="OK"/>
2290 <TestResult>true</TestResult>
2291 </TestMembershipResponse>
2292
2293
```

2294 12. Confident that Bob is a member of the group against which Alice defined privileges, SPa grants Bob access to the
2295 resource in question.

2296 5.3. Group Operation

2297 The following demonstrates the sequence of steps and messages when a user desires that some operation (e.g. send an
2298 invitation) be performed on members of a particular group in their PS list.

2299 1. Alice signs on to SPa.

2300 2. Alice requests that SPa sends a party invitation to all members in a group.

2301 3. SPa/WSCa finds PSa, via DSa.

2302 4. SPa/WSCa queries PSa for the list of available groups and displays to Alice. Alice picks the relevant group on
2303 whose members she wishes to operate. SPa/WSCa requests from PSa a list of members of the specified group.

```
2304 <ListMembersRequest>  
2305 <TargetObjectID>https://ps.com/nmerflas</TargetObjectID>  
2306 </ListMembersRequest>  
2307  
2308
```

2309 5. PSa responds with a list of members to SPa/WSCa.

```
2310 <ListMembersResponse>  
2311 <Status code="OK"/>  
2312 <Object NodeType="urn:liberty:ps:entity" >  
2313 <ObjectID="https://psa.com/sdfhgusfsf"/>  
2314 <DisplayName>Bob</DisplayName>  
2315 </Object>  
2316 <Object NodeType="urn:liberty:ps:entity">  
2317 <ObjectID="https://psa.com/itndojd"/>  
2318 <DisplayName>Mary</DisplayName>  
2319 </Object>  
2320 </ListMembersResponse>  
2321  
2322
```

2323 6. SPa/WSCa sends a ResolveIdentifierRequest messages with appropriate TargetObjectID elements to
2324 PSa to request identity tokens for Bob & Mary.

2325 Note: For sake of demonstration, we assume here that by chance Bob & Mary share the same IDP but this will
2326 not be the general case.

```
2327 <ResolveIdentifierRequest>  
2328 <ResolveInput reqID="0">  
2329 <TargetObjectID>https://psa.com/sdfhgusfsf</TargetObjectID> <!-- Bob -->  
2330 </ResolveInput>  
2331 <ResolveInput reqID="1">  
2332 <TargetObjectID>https://psa.com/itndojd</TargetObjectID> <!-- Alice -->  
2333 </ResolveInput>  
2334 </ResolveIdentifierRequest>  
2335  
2336
```

2337 7. PSa sends a `ims:IdentityMappingRequest` message to IDPb including the existing identity token between
2338 PSa and IDPb, specifying SPa as the target provider.

```
2339 <ims:IdentityMappingRequest>
2340 <ims:MappingInput reqID="2">
2341 <sec:TokenPolicy>
2342 <samlp:NameIDPolicy SPNameQualifier="https://spa.com"/>
2343 </sec:TokenPolicy>
2344 <sec:Token>
2345 <saml:Assertion>
2346 existing identity token for Bob between PSa and IDPb
2347 </saml:Assertion>
2348 </sec:Token>
2349 </ims:MappingInput>
2350 <ims:MappingInput reqID="3">
2351 <sec:TokenPolicy>
2352 <samlp:NameIDPolicy SPNameQualifier="https://spa.com"/>
2353 </sec:TokenPolicy>
2354 <sec:Token>
2355 <saml:Assertion>
2356 existing identity token for Alice between PSa and IDPb
2357 </saml:Assertion>
2358 </sec:Token>
2359 </ims:MappingInput>
2360 </ims:IdentityMappingRequest>
2361
2362
```

2363 8. IDPb sends an `ims:IdentityMappingResponse` message with identity tokens for Bob and Mary between SPa
2364 and IDPb.

```
2365 <ims:IdentityMappingResponse>
2366 <ims:MappingOutput reqRef="2">
2367 <Status>OK</Status>
2368 <sec:Token>
2369 <saml:Assertion>
2370 an identity token for Bob in SPa/WSCa's namespace goes here
2371 </saml:Assertion>
2372 </sec:Token>
2373 </ims:MappingOutput>
2374 <ims:MappingOutput reqRef="3">
2375 <Status>OK</Status>
2376 <sec:Token>
2377 <saml:Assertion>
2378 an identity token for Alice in SPa/WSCa's namespace goes here
2379 </saml:Assertion>
2380 </sec:Token>
2381 </ims:MappingOutput>
2382 </ims:IdentityResponse>
2383
2384
```

2385 9. PSa forwards on the identity tokens to SPa/WSCa in its `ResolveIdentifierResponse` message to the original
2386 `ResolveIdentifierRequest` message from SPa/WSCa.

```
2387 <ResolveIdentifierResponse>
2388 <Status code="OK"/>
2389 <ResolveOutput reqRef="0">
2390 <Token>
2391 <saml:Assertion>
2392 an identity token for Bob in SPa/WSCa's namespace goes here
2393 </saml:Assertion>
2394 </Token>
2395 </ResolveOutput>
2396 <ResolveOutput reqRef="1">
```

```
2398     <Token>
2399       <saml:Assertion>
2400         an identity token for Alice in SPa/WSCa's namespace goes here
2401       </saml:Assertion>
2402     </Token>
2403   </ResolveOutput>
2404 </ResolveIdentifierResponse>
2405
```

2406 10. Once SPa/WSCa has the identity tokens for Bob & Alice, it is able to use the embedded bootstrap for Discovery
2407 Services to discover relevant WSPs, e.g. a Personal Profile service so as to get email addresses in order to send
2408 the party invitation

2409 6. Security Considerations

2410 A discussion of security considerations unique to the People Service and the user interaction model.

2411 • The header blocks specified in this document should be integrity-protected using the mechanisms detailed in
2412 [[LibertySecMech](#)].

2413 • Header blocks should be signed in accordance with [[LibertySecMech](#)]. The receiver of a message containing a
2414 signature that covers specific header blocks should verify the signature as part of verifying the integrity of the
2415 header block.

2416 • Metadata [[LibertyMetadata](#)] should be used to the greatest extent possible to verify message sender identity claims.

2417 • Message senders and receivers should be authenticated to one another via the mechanisms discussed in [[Liberty-](#)
2418 [SecMech](#)].

2419 7. XML Schema for ID-WSF People Service

2420 The formal XML schema for the ID-WSF People Service follows:

```
2421
2422 <xs:schema
2423   targetNamespace="urn:liberty:ps:2006-08"
2424   xmlns="urn:liberty:ps:2006-08"
2425   xmlns:lu="urn:liberty:util:2006-08"
2426   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2427   xmlns:ims="urn:liberty:ims:2006-08"
2428   xmlns:subs="urn:liberty:ssos:2006-08"
2429   xmlns:sec="urn:liberty:security:2006-08"
2430   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
2431   elementFormDefault="qualified"
2432   attributeFormDefault="unqualified">
2433
2434   <xs:import namespace="urn:liberty:util:2006-08" schemaLocation="liberty-idwsf-utility-v2.0.xsd"
2435   />
2436   <xs:import namespace="urn:liberty:ims:2006-08" schemaLocation="liberty-idwsf-idmap
2437 ping-svc-v2.0.xsd" />
2438   <xs:import namespace="urn:liberty:ssos:2006-08" schemaLocation="liberty-idwsf-subsv1.0.xsd" />
2439   <xs:import namespace="urn:liberty:security:2006-08" schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd" />
2440   <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol" schemaLocation="saml-schema-protocol-2.0.xsd" />
2441
2442   <xs:annotation>
2443     <xs:documentation>
2444       The source code in this XSD file was excerpted verbatim from:
2445       Liberty ID-WSF People Service Specification
2446       Version 1.0 Draft
2447       30 July, 2006
2448       Copyright (c) 2006 Liberty Alliance participants, see
2449       http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.html
2450     </xs:documentation>
2451   </xs:annotation>
2452
2453   <!-- Definition of LocalizedDisplayNameType -->
2454   <xs:complexType name="LocalizedDisplayNameType">
2455     <xs:simpleContent>
2456       <xs:extension base="xs:string">
2457         <xs:attribute name="Locale" type="xs:language" use="required"/>
2458         <xs:attribute name="IsDefault" type="xs:boolean" use="optional"/>
2459       </xs:extension>
2460     </xs:simpleContent>
2461   </xs:complexType>
2462
2463   <!-- Definition of TagType -->
2464   <xs:complexType name="TagType">
2465     <xs:simpleContent>
2466       <xs:extension base="xs:string">
2467         <xs:attribute name="Ref" type="xs:anyURI" use="required"/>
2468       </xs:extension>
2469     </xs:simpleContent>
2470   </xs:complexType>
2471
2472   <!-- Declaration of ObjectID element -->
2473   <xs:element name="ObjectID" type="ObjectIDType"/>
2474
2475   <!-- Declaration of TargetObjectID element -->
2476   <xs:element name="TargetObjectID" type="ObjectIDType"/>
2477
2478   <!-- Definition of ObjectIDType -->
2479   <xs:complexType name="ObjectIDType">
```

```
2484     <xs:simpleContent>
2485       <xs:extension base="xs:anyURI" />
2486     </xs:simpleContent>
2487   </xs:complexType>
2488
2489   <!-- Declaration of Object element -->
2490   <xs:element name="Object" type="ObjectType"/>
2491
2492   <!-- Definition of ObjectType -->
2493   <xs:complexType name="ObjectType">
2494     <xs:sequence>
2495       <xs:element ref="ObjectID" minOccurs="0" />
2496       <xs:element name="DisplayName" type="LocalizedDisplayNameType" minOccurs="1" maxOccurs="unbounded" />
2497
2498       <xs:element name="Tag" type="TagType" minOccurs="0" />
2499       <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded" />
2500       <xs:element name="ObjectRef" type="ObjectIDType" minOccurs="0" maxOccurs="unbounded" />
2501     </xs:sequence>
2502     <xs:attribute name="NodeType" type="xs:anyURI" use="required" />
2503     <xs:attribute name="CreatedDateTime" type="xs:dateTime" use="optional" />
2504     <xs:attribute name="ModifiedDateTime" type="xs:dateTime" use="optional" />
2505   </xs:complexType>
2506
2507   <!-- Declaration of PStoSPRedirectURL-->
2508
2509   <xs:element name="PStoSPRedirectURL" type="PStoSPRedirectURLType"/>
2510
2511   <!-- Definition of PStoSPRedirectURLType-->
2512
2513   <xs:complexType name="PStoSPRedirectURLType">
2514     <xs:annotation>
2515       <xs:documentation>When sending a AddEntityRequest to a PS provider, the SP may insert a PStoSPRedirectURL. It will be to this U
2516     </xs:annotation>
2517     <xs:simpleContent>
2518       <xs:extension base="xs:anyURI" />
2519     </xs:simpleContent>
2520   </xs:complexType>
2521
2522   <!-- Declaration of SptoPSRedirectURL-->
2523
2524   <xs:element name="SptoPSRedirectURL" type="SptoPSRedirectURLType"/>
2525
2526   <!-- Definition of SptoPSRedirectURLType-->
2527
2528   <xs:complexType name="SptoPSRedirectURLType">
2529     <xs:annotation>
2530       <xs:documentation>A PS provider may insert a SptoPSRedirectURL in its AddEntityResponse. It will be to this U
2531     </xs:annotation>
2532     <xs:simpleContent>
2533       <xs:extension base="xs:anyURI" />
2534     </xs:simpleContent>
2535   </xs:complexType>
2536
2537   <!-- Declaration of QueryString -->
2538
2539   <xs:element name="QueryString" type="QueryStringType"/>
2540
2541   <!-- Definition of QueryStringType-->
2542
2543   <xs:complexType name="QueryStringType">
2544     <xs:annotation>
2545       <xs:documentation>A PS provider may insert a QueryString in its AddEntityResponse or AddKnownEntityResponse.
2546     </xs:annotation>
2547     <xs:simpleContent>
2548       <xs:extension base="xs:string" />
2549     </xs:simpleContent>
2550   </xs:complexType>
```

```
2551         <xs:extension base="xs:string" />
2552     </xs:simpleContent>
2553 </xs:complexType>
2554
2555 <!-- Declaration of CreatePSObject element -->
2556 <xs:element name="CreatePSObject" />
2557
2558 <!-- Definition of RequestAbstractType -->
2559 <xs:complexType name="RequestAbstractType" abstract="true">
2560     <xs:anyAttribute namespace="##other" processContents="lax" />
2561 </xs:complexType>
2562
2563 <!-- Definition of ResponseAbstractType -->
2564 <xs:complexType name="ResponseAbstractType" abstract="true">
2565     <xs:sequence>
2566         <xs:element ref="lu:Status" />
2567     </xs:sequence>
2568     <xs:anyAttribute namespace="##other" processContents="lax" />
2569 </xs:complexType>
2570
2571 <!-- Declaration of AddEntityRequest element -->
2572 <xs:element name="AddEntityRequest" type="AddEntityRequestType" />
2573 <!-- Definition of AddEntityRequestType -->
2574 <xs:complexType name="AddEntityRequestType">
2575     <xs:complexContent>
2576         <xs:extension base="RequestAbstractType">
2577             <xs:sequence>
2578                 <xs:element ref="Object" />
2579                 <xs:element ref="PStoSPRedirectURL" minOccurs="0" />
2580                 <xs:element ref="CreatePSObject" minOccurs="0" />
2581                 <xs:element ref="Subscription" minOccurs="0" />
2582                 <xs:element ref="sec:TokenPolicy" minOccurs="0" />
2583             </xs:sequence>
2584         </xs:extension>
2585     </xs:complexContent>
2586 </xs:complexType>
2587
2588 <!-- Declaration of AddEntityResponse element -->
2589 <xs:element name="AddEntityResponse" type="AddEntityResponseType" />
2590 <!-- Definition of AddEntityResponseType -->
2591 <xs:complexType name="AddEntityResponseType">
2592     <xs:complexContent>
2593         <xs:extension base="ResponseAbstractType">
2594             <xs:sequence>
2595                 <xs:element ref="Object" minOccurs="0" />
2596                 <xs:element ref="SPtoPSRedirectURL" minOccurs="0" maxOccurs="1" />
2597                 <xs:element ref="QueryString" minOccurs="0" maxOccurs="1" />
2598             </xs:sequence>
2599         </xs:extension>
2600     </xs:complexContent>
2601 </xs:complexType>
2602
2603 <!-- Declaration of AddKnownEntityRequest element -->
2604 <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType" />
2605 <!-- Definition of AddKnownEntityRequestType -->
2606 <xs:complexType name="AddKnownEntityRequestType">
2607     <xs:complexContent>
2608         <xs:extension base="RequestAbstractType">
2609             <xs:sequence>
2610                 <xs:element ref="Object" />
2611                 <xs:element ref="sec:Token" />
2612                 <xs:element ref="CreatePSObject" minOccurs="0" />
2613                 <xs:element ref="Subscription" minOccurs="0" />
2614                 <xs:element ref="sec:TokenPolicy" minOccurs="0" />
2615             </xs:sequence>
2616         </xs:extension>
2617     </xs:complexContent>
```

```
2618     </xs:complexType>
2619
2620     <!-- Declaration of AddKnownEntityResponse element -->
2621     <xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType"/>
2622     <!-- Definition of AddKnownEntityResponseType -->
2623     <xs:complexType name="AddKnownEntityResponseType">
2624         <xs:complexContent>
2625             <xs:extension base="ResponseAbstractType">
2626                 <xs:sequence>
2627                     <xs:element ref="Object" minOccurs="0"/>
2628                     <xs:element ref="SPtoPSRedirectURL" minOccurs="0" maxOccurs="1"/>
2629                     <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
2630                 </xs:sequence>
2631             </xs:extension>
2632         </xs:complexContent>
2633     </xs:complexType>
2634
2635     <!-- Declaration of AddCollectionRequest element -->
2636     <xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
2637     <!-- Definition of AddCollectionRequestType -->
2638     <xs:complexType name="AddCollectionRequestType">
2639         <xs:complexContent>
2640             <xs:extension base="RequestAbstractType">
2641                 <xs:sequence>
2642                     <xs:element ref="Object"/>
2643                     <xs:element ref="Subscription" minOccurs="0"/>
2644                 </xs:sequence>
2645             </xs:extension>
2646         </xs:complexContent>
2647     </xs:complexType>
2648
2649     <!-- Declaration of AddCollectionResponse element -->
2650     <xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>
2651     <!-- Definition of AddCollectionResponseType -->
2652     <xs:complexType name="AddCollectionResponseType">
2653         <xs:complexContent>
2654             <xs:extension base="ResponseAbstractType">
2655                 <xs:sequence>
2656                     <xs:element ref="Object" minOccurs="0"/>
2657                 </xs:sequence>
2658             </xs:extension>
2659         </xs:complexContent>
2660     </xs:complexType>
2661
2662     <!-- Declaration of AddToCollectionRequest element -->
2663     <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>
2664     <!-- Definition of AddToCollectionRequestType -->
2665     <xs:complexType name="AddToCollectionRequestType">
2666         <xs:complexContent>
2667             <xs:extension base="RequestAbstractType">
2668                 <xs:sequence>
2669                     <xs:element ref="TargetObjectID"/>
2670                     <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>
2671                     <xs:element ref="Subscription" minOccurs="0"/>
2672                 </xs:sequence>
2673             </xs:extension>
2674         </xs:complexContent>
2675     </xs:complexType>
2676
2677     <!-- Declaration of AddToCollectionResponse element -->
2678     <xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>
2679
2680     <!-- Declaration of RemoveEntityRequest element -->
2681     <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType"/>
2682     <!-- Definition of RemoveEntityRequestType -->
2683     <xs:complexType name="RemoveEntityRequestType">
2684         <xs:complexContent>
```

```
2685         <xs:extension base="RequestAbstractType">
2686             <xs:sequence>
2687                 <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
2688             </xs:sequence>
2689         </xs:extension>
2690     </xs:complexContent>
2691 </xs:complexType>
2692
2693 <!-- Declaration of RemoveEntityResponse element -->
2694 <xs:element name="RemoveEntityResponse" type="ResponseAbstractType"/>
2695
2696 <!-- Declaration of RemoveCollectionRequest element -->
2697 <xs:element name="RemoveCollectionRequest" type="RemoveCollectionRequestType"/>
2698 <!-- Definition of RemoveCollectionRequestType -->
2699 <xs:complexType name="RemoveCollectionRequestType">
2700     <xs:complexContent>
2701         <xs:extension base="RequestAbstractType">
2702             <xs:sequence>
2703                 <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
2704             </xs:sequence>
2705         </xs:extension>
2706     </xs:complexContent>
2707 </xs:complexType>
2708
2709 <!-- Declaration of RemoveCollectionResponse element -->
2710 <xs:element name="RemoveCollectionResponse" type="ResponseAbstractType"/>
2711
2712 <!-- Declaration of RemoveFromCollectionRequest element -->
2713 <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
2714 <!-- Definition of RemoveFromCollectionRequestType -->
2715 <xs:complexType name="RemoveFromCollectionRequestType">
2716     <xs:complexContent>
2717         <xs:extension base="RequestAbstractType">
2718             <xs:sequence>
2719                 <xs:element ref="TargetObjectID"/>
2720                 <xs:element ref="ObjectID" maxOccurs="unbounded"/>
2721                 <xs:element ref="Subscription" minOccurs="0"/>
2722             </xs:sequence>
2723         </xs:extension>
2724     </xs:complexContent>
2725 </xs:complexType>
2726
2727 <!-- Declaration of RemoveFromCollectionResponse element -->
2728 <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>
2729
2730 <!-- Declaration of ListMembersRequest element -->
2731 <xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
2732 <!-- Definition of ListMembersRequestType -->
2733 <xs:complexType name="ListMembersRequestType">
2734     <xs:complexContent>
2735         <xs:extension base="RequestAbstractType">
2736             <xs:sequence>
2737                 <xs:element ref="TargetObjectID" minOccurs="0"/>
2738                 <xs:element ref="Subscription" minOccurs="0"/>
2739             </xs:sequence>
2740             <xs:attribute name="Structured" type="xs:anyURI" use="optional"/>
2741             <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
2742             <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
2743         </xs:extension>
2744     </xs:complexContent>
2745 </xs:complexType>
2746
2747 <!-- Declaration of ListMembersResponse element -->
2748 <xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
2749 <!-- Definition of ListMembersResponseType -->
2750 <xs:complexType name="ListMembersResponseType">
2751     <xs:complexContent>
```

```
2752         <xs:extension base="ResponseAbstractType">
2753             <xs:sequence>
2754                 <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2755             </xs:sequence>
2756         </xs:extension>
2757     </xs:complexContent>
2758 </xs:complexType>
2759
2760 <!-- Declaration of QueryObjectsRequest element -->
2761 <xs:element name="QueryObjectsRequest" type="QueryObjectsRequestType"/>
2762 <!-- Definition of QueryObjectsRequestType -->
2763 <xs:complexType name="QueryObjectsRequestType" >
2764     <xs:complexContent>
2765         <xs:extension base="RequestAbstractType">
2766             <xs:sequence>
2767                 <xs:element name="Filter" type="xs:string"/>
2768                 <xs:element ref="Subscription" minOccurs="0"/>
2769             </xs:sequence>
2770             <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
2771             <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
2772         </xs:extension>
2773     </xs:complexContent>
2774 </xs:complexType>
2775
2776 <!-- Declaration of QueryObjectsResponse element -->
2777 <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
2778 <!-- Definition of QueryObjectsResponseType -->
2779 <xs:complexType name="QueryObjectsResponseType">
2780     <xs:complexContent>
2781         <xs:extension base="ResponseAbstractType">
2782             <xs:sequence>
2783                 <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2784             </xs:sequence>
2785         </xs:extension>
2786     </xs:complexContent>
2787 </xs:complexType>
2788
2789 <!-- Declaration of GetObjectInfoRequest element -->
2790 <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
2791 <!-- Definition of GetObjectInfoRequestType -->
2792 <xs:complexType name="GetObjectInfoRequestType">
2793     <xs:complexContent>
2794         <xs:extension base="RequestAbstractType">
2795             <xs:sequence>
2796                 <xs:element ref="TargetObjectID" minOccurs="0"/>
2797                 <xs:element ref="Subscription" minOccurs="0"/>
2798             </xs:sequence>
2799         </xs:extension>
2800     </xs:complexContent>
2801 </xs:complexType>
2802
2803 <!-- Declaration of GetObjectInfoResponse element -->
2804 <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
2805 <!-- Definition of GetObjectInfoResponseType -->
2806 <xs:complexType name="GetObjectInfoResponseType">
2807     <xs:complexContent>
2808         <xs:extension base="ResponseAbstractType">
2809             <xs:sequence>
2810                 <xs:element ref="Object" minOccurs="0"/>
2811             </xs:sequence>
2812         </xs:extension>
2813     </xs:complexContent>
2814 </xs:complexType>
2815
2816 <!-- Declaration of SetObjectInfoRequest element -->
2817 <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
2818 <!-- Definition of SetObjectInfoRequestType -->
```

```
2819 <xs:complexType name="SetObjectInfoRequestType">
2820 <xs:complexContent>
2821 <xs:extension base="RequestAbstractType">
2822 <xs:sequence>
2823 <xs:element ref="Object" maxOccurs="unbounded"/>
2824 <xs:element ref="Subscription" minOccurs="0"/>
2825 </xs:sequence>
2826 </xs:extension>
2827 </xs:complexContent>
2828 </xs:complexType>
2829
2830 <!-- Declaration of SetObjectInfoResponse element -->
2831 <xs:element name="SetObjectInfoResponse" type="ResponseAbstractType"/>
2832
2833 <!-- Declaration of TestMembershipRequest element -->
2834 <xs:element name="TestMembershipRequest" type="TestMembershipRequestType" />
2835 <!-- Definition of TestMembershipRequestType -->
2836 <xs:complexType name="TestMembershipRequestType">
2837 <xs:complexContent>
2838 <xs:extension base="RequestAbstractType">
2839 <xs:sequence>
2840 <xs:element ref="TargetObjectID" minOccurs="0"/>
2841 <xs:element ref="sec:Token"/>
2842 <xs:element ref="Subscription" minOccurs="0"/>
2843 </xs:sequence>
2844 </xs:extension>
2845 </xs:complexContent>
2846 </xs:complexType>
2847
2848 <!-- Definition of ResultType -->
2849 <xs:complexType name="ResultType">
2850 <xs:simpleContent>
2851 <xs:extension base="xs:boolean"/>
2852 </xs:simpleContent>
2853 </xs:complexType>
2854
2855 <!-- Declaration of TestMembershipResponse element -->
2856 <xs:element name="TestMembershipResponse" type="TestMembershipResponseType"/>
2857 <!-- Definition of TestMembershipResponseType -->
2858 <xs:complexType name="TestMembershipResponseType">
2859 <xs:complexContent>
2860 <xs:extension base="ResponseAbstractType">
2861 <xs:sequence>
2862 <xs:element name="Result" type="ResultType" minOccurs="0"/>
2863 </xs:sequence>
2864 </xs:extension>
2865 </xs:complexContent>
2866 </xs:complexType>
2867
2868 <!-- Declaration of ResolveIdentifierRequest element -->
2869 <xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
2870 <!-- Definition of ResolveIdentifierRequestType -->
2871 <xs:complexType name="ResolveIdentifierRequestType">
2872 <xs:complexContent>
2873 <xs:extension base="RequestAbstractType">
2874 <xs:sequence>
2875 <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
2876 </xs:sequence>
2877 </xs:extension>
2878 </xs:complexContent>
2879 </xs:complexType>
2880
2881 <!-- Declaration of ResolveInput element -->
2882 <xs:element name="ResolveInput" type="ResolveInputType"/>
2883 <!-- Definition of ResolveInputType -->
2884 <xs:complexType name="ResolveInputType">
2885 <xs:complexContent>
```

```
2886         <xs:extension base="ims:MappingInputType">
2887             <xs:sequence>
2888                 <xs:element ref="TargetObjectID" minOccurs="0" />
2889             </xs:sequence>
2890         </xs:extension>
2891     </xs:complexContent>
2892 </xs:complexType>
2893
2894 <!-- Declaration of ResolveIdentifierResponse element -->
2895 <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType" />
2896 <!-- Definition of ResolveIdentifierResponseType -->
2897 <xs:complexType name="ResolveIdentifierResponseType" >
2898     <xs:complexContent>
2899         <xs:extension base="ResponseAbstractType">
2900             <xs:sequence>
2901                 <xs:element ref="ResolveOutput" maxOccurs="unbounded" />
2902             </xs:sequence>
2903         </xs:extension>
2904     </xs:complexContent>
2905 </xs:complexType>
2906
2907 <!-- Declaration of ResolveOutput element -->
2908 <xs:element name="ResolveOutput" type="ims:MappingOutputType" />
2909
2910 <!-- Declaration of Subscription element -->
2911 <xs:element name="Subscription" type="subs:SubscriptionType" />
2912
2913 <!-- Declaration of Notification element -->
2914 <xs:element name="Notification" type="NotificationType" />
2915 <!-- Definition of NotificationType -->
2916 <xs:complexType name="NotificationType">
2917     <xs:complexContent>
2918         <xs:extension base="subs:NotificationType">
2919             <xs:sequence>
2920                 <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded" />
2921             </xs:sequence>
2922         </xs:extension>
2923     </xs:complexContent>
2924 </xs:complexType>
2925
2926 <!-- Declaration of ItemData element -->
2927 <xs:element name="ItemData" type="ItemDataType" />
2928 <!-- Definition of ItemDataType -->
2929 <xs:complexType name="ItemDataType">
2930     <xs:sequence>
2931         <xs:element ref="Object" />
2932     </xs:sequence>
2933 </xs:complexType>
2934
2935 <!-- Declaration of Notify element -->
2936 <xs:element name="Notify" type="NotifyType" />
2937 <!-- Definition of NotifyType -->
2938 <xs:complexType name="NotifyType">
2939     <xs:complexContent>
2940         <xs:extension base="RequestAbstractType">
2941             <xs:sequence>
2942                 <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />
2943             </xs:sequence>
2944             <xs:attributeGroup ref="subs:NotifyAttributeGroup" />
2945         </xs:extension>
2946     </xs:complexContent>
2947 </xs:complexType>
2948
2949 <!-- Declaration of NotifyResponse element -->
2950 <xs:element name="NotifyResponse" type="subs:NotifyResponseType" />
2951
2952 </xs:schema>
```

2953
2954

2955 **8. Abstract WSDL**

```
2956
2957 <definitions
2958   name="id-wsf-ps_2006-08_wsd_interface"
2959   targetNamespace="urn:liberty:ps:2006-08"
2960   xmlns:tns="urn:liberty:ps:2006-08"
2961   xmlns="http://schemas.xmlsoap.org/wsd/"
2962   xmlns:soap="http://schemas.xmlsoap.org/wsd/soap/"
2963   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2964   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsd"
2965   xmlns:ps="urn:liberty:ps:2006-08"
2966   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2967   xsi:schemaLocation="http://schemas.xmlsoap.org/wsd/
2968     http://schemas.xmlsoap.org/wsd/
2969     http://www.w3.org/2006/02/addressing/wsd
2970     http://www.w3.org/2006/02/addressing/wsd/ws-addr-wsd.xsd">
2971
2972   <xsd:documentation>
2973
2974     XML Schema from Liberty People Service Specification.
2975
2976     ### NOTICE ###
2977
2978     Copyright (c) 2006 Liberty Alliance participants, see
2979     http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
2980
2981   </xsd:documentation>
2982
2983   <types>
2984     <xsd:schema>
2985       <xsd:import namespace="urn:liberty:ps:2006-08"
2986         schemaLocation="liberty-idwsf-people-service-v1.0.xsd"/>
2987     </xsd:schema>
2988   </types>
2989
2990   <!-- Messages -->
2991
2992   <!-- Adding a User -->
2993
2994   <message name="AddEntityRequest">
2995     <part name="body" element="ps:AddEntityRequest"/>
2996   </message>
2997
2998   <message name="AddEntityResponse">
2999     <part name="body" element="ps:AddEntityResponse"/>
3000   </message>
3001
3002   <!-- Adding a Known User -->
3003
3004   <message name="AddKnownEntityRequest">
3005     <part name="body" element="ps:AddKnownEntityRequest"/>
3006   </message>
3007
3008   <message name="AddKnownEntityResponse">
3009     <part name="body" element="ps:AddKnownEntityResponse"/>
3010   </message>
3011
3012   <!-- Removing a User -->
3013
3014   <message name="RemoveEntityRequest">
3015     <part name="body" element="ps:RemoveEntityRequest"/>
3016   </message>
3017
3018   <message name="RemoveEntityResponse">
3019     <part name="body" element="ps:RemoveEntityResponse"/>
3020
```

```
3021 </message>
3022
3023 <!-- Adding a Group -->
3024
3025 <message name="AddCollectionRequest">
3026   <part name="body" element="ps:AddCollectionRequest" />
3027 </message>
3028
3029 <message name="AddCollectionResponse">
3030   <part name="body" element="ps:AddCollectionResponse" />
3031 </message>
3032
3033 <!-- Removing a Group -->
3034
3035 <message name="RemoveCollectionRequest">
3036   <part name="body" element="ps:RemoveCollectionRequest" />
3037 </message>
3038
3039 <message name="RemoveCollectionResponse">
3040   <part name="body" element="ps:RemoveCollectionResponse" />
3041 </message>
3042
3043 <!-- Adding to a Group -->
3044
3045 <message name="AddToCollectionRequest">
3046   <part name="body" element="ps:AddToCollectionRequest" />
3047 </message>
3048
3049 <message name="AddToCollectionResponse">
3050   <part name="body" element="ps:AddToCollectionResponse" />
3051 </message>
3052
3053 <!-- Removing From a Group -->
3054
3055 <message name="RemoveFromCollectionRequest">
3056   <part name="body" element="ps:RemoveFromCollectionRequest" />
3057 </message>
3058
3059 <message name="RemoveFromCollectionResponse">
3060   <part name="body" element="ps:RemoveFromCollectionResponse" />
3061 </message>
3062
3063 <!-- Listing Members -->
3064
3065 <message name="ListMembersRequest">
3066   <part name="body" element="ps:ListMembersRequest" />
3067 </message>
3068
3069 <message name="ListMembersResponse">
3070   <part name="body" element="ps:ListMembersResponse" />
3071 </message>
3072
3073 <!-- Retrieving Object Info -->
3074
3075 <message name="GetObjectInfoRequest">
3076   <part name="body" element="ps:GetObjectInfoRequest" />
3077 </message>
3078
3079 <message name="GetObjectInfoResponse">
3080   <part name="body" element="ps:GetObjectInfoResponse" />
3081 </message>
3082
3083 <!-- Updating Object Info -->
3084
3085 <message name="SetObjectInfoRequest">
3086   <part name="body" element="ps:SetObjectInfoRequest" />
3087 </message>
```

```
3088
3089     <message name="SetObjectInfoResponse">
3090         <part name="body" element="ps:SetObjectInfoResponse" />
3091     </message>
3092
3093     <!-- Querying Objects -->
3094
3095     <message name="QueryObjectsRequest">
3096         <part name="body" element="ps:QueryObjectsRequest" />
3097     </message>
3098
3099     <message name="QueryObjectsResponse">
3100         <part name="body" element="ps:QueryObjectsResponse" />
3101     </message>
3102
3103     <!-- Testing Membership -->
3104
3105     <message name="TestMembershipRequest">
3106         <part name="body" element="ps:TestMembershipRequest" />
3107     </message>
3108
3109     <message name="TestMembershipResponse">
3110         <part name="body" element="ps:TestMembershipResponse" />
3111     </message>
3112
3113     <!-- Resolving Identifiers -->
3114
3115     <message name="ResolveIdentifierRequest">
3116         <part name="body" element="ps:ResolveIdentifierRequest" />
3117     </message>
3118
3119     <message name="ResolveIdentifierResponse">
3120         <part name="body" element="ps:ResolveIdentifierResponse" />
3121     </message>
3122
3123     <!-- Port Type -->
3124
3125     <portType name="LibertyPS1">
3126
3127         <operation name="AddEntity">
3128             <input message="tns:AddEntityRequest"
3129 wsaw:Action="urn:liberty:ps:2006-08:AddEntityRequest" />
3130             <output message="tns:AddEntityResponse"
3131 wsaw:Action="urn:liberty:ps:2006-08:AddEntityResponse" />
3132         </operation>
3133
3134         <operation name="AddKnownEntity">
3135             <input message="tns:AddKnownEntityRequest"
3136 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityRequest" />
3137             <output message="tns:AddKnownEntityResponse"
3138 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityResponse" />
3139         </operation>
3140
3141         <operation name="RemoveEntity">
3142             <input message="tns:RemoveEntityRequest"
3143 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityRequest" />
3144             <output message="tns:RemoveEntityResponse"
3145 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityResponse" />
3146         </operation>
3147
3148         <operation name="AddCollection">
3149             <input message="tns:AddCollectionRequest"
3150 wsaw:Action="urn:liberty:ps:2006-08:AddCollectionRequest" />
3151             <output message="tns:AddCollectionResponse"
3152 wsaw:Action="urn:liberty:ps:2006-08:AddCollectionResponse" />
3153         </operation>
3154
```

```
3155     <operation name="RemoveCollection">
3156         <input message="tns:RemoveCollectionRequest"
3157 wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionRequest" />
3158         <output message="tns:RemoveCollectionResponse"
3159 wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionResponse" />
3160     </operation>
3161
3162     <operation name="AddToCollection">
3163         <input message="tns:AddToCollectionRequest"
3164 wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionRequest" />
3165         <output message="tns:AddToCollectionResponse"
3166 wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionResponse" />
3167     </operation>
3168
3169     <operation name="RemoveFromCollection">
3170         <input message="tns:RemoveFromCollectionRequest"
3171 wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionRequest" />
3172         <output message="tns:RemoveFromCollectionResponse"
3173 wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionResponse" />
3174     </operation>
3175
3176     <operation name="ListMembersOfCollection">
3177         <input message="tns:ListMembersRequest"
3178 wsaw:Action="urn:liberty:ps:2006-08:ListMembersRequest" />
3179         <output message="tns:ListMembersResponse"
3180 wsaw:Action="urn:liberty:ps:2006-08:ListMembersResponse" />
3181     </operation>
3182
3183     <operation name="GetObjectInfo">
3184         <input message="tns:GetObjectInfoRequest"
3185 wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoRequest" />
3186         <output message="tns:GetObjectInfoResponse"
3187 wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoResponse" />
3188     </operation>
3189
3190     <operation name="SetObjectInfo">
3191         <input message="tns:SetObjectInfoRequest"
3192 wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoRequest" />
3193         <output message="tns:SetObjectInfoResponse"
3194 wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoResponse" />
3195     </operation>
3196
3197     <operation name="QueryObjects">
3198         <input message="tns:QueryObjectsRequest"
3199 wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsRequest" />
3200         <output message="tns:QueryObjectsResponse"
3201 wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsResponse" />
3202     </operation>
3203
3204     <operation name="TestMembership">
3205         <input message="tns:TestMembershipRequest"
3206 wsaw:Action="urn:liberty:ps:2006-08:TestMembershipRequest" />
3207         <output message="tns:TestMembershipResponse"
3208 wsaw:Action="urn:liberty:ps:2006-08:TestMembershipResponse" />
3209     </operation>
3210
3211     <operation name="ResolveIdentifier">
3212         <input message="tns:ResolveIdentifierRequest"
3213 wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierRequest" />
3214         <output message="tns:ResolveIdentifierResponse"
3215 wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierResponse" />
3216     </operation>
3217
3218 </portType>
3219
3220 <!--
3221 An example of a binding and service that can be used with this
```

```
3222 abstract service description is provided below.
3223 -->
3224
3225 <binding name="PeopleServiceSoapBinding" type="tns:LibertyPS1">
3226
3227 <soap:binding style="document"
3228 transport="http://schemas.xmlsoap.org/soap/http" />
3229
3230 <operation name="AddEntity">
3231 <soap:operation soapAction="urn:liberty:ps:2006-08:AddEntityRequest" />
3232 <input> <soap:body use="literal" /> </input>
3233 <output> <soap:body use="literal" /> </output>
3234 </operation>
3235
3236 <operation name="AddKnownEntity">
3237 <soap:operation soapAction="urn:liberty:ps:2006-08:AddKnownEntityRequest" />
3238 <input> <soap:body use="literal" /> </input>
3239 <output> <soap:body use="literal" /> </output>
3240 </operation>
3241 <operation name="RemoveEntity">
3242 <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveEntityRequest" />
3243 <input> <soap:body use="literal" /> </input>
3244 <output> <soap:body use="literal" /> </output>
3245 </operation>
3246 <operation name="AddCollection">
3247 <soap:operation soapAction="urn:liberty:ps:2006-08:AddCollectionRequest" />
3248 <input> <soap:body use="literal" /> </input>
3249 <output> <soap:body use="literal" /> </output>
3250 </operation>
3251 <operation name="RemoveCollection">
3252 <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveCollectionRequest" />
3253 <input> <soap:body use="literal" /> </input>
3254 <output> <soap:body use="literal" /> </output>
3255 </operation>
3256 <operation name="AddToCollection">
3257 <soap:operation soapAction="urn:liberty:ps:2006-08:AddToCollectionRequest" />
3258 <input> <soap:body use="literal" /> </input>
3259 <output> <soap:body use="literal" /> </output>
3260 </operation>
3261 <operation name="RemoveFromCollection">
3262 <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveFromCollectionRequest" />
3263 <input> <soap:body use="literal" /> </input>
3264 <output> <soap:body use="literal" /> </output>
3265 </operation>
3266 <operation name="ListMembersOfCollection">
3267 <soap:operation soapAction="urn:liberty:ps:2006-08:ListMembersOfCollectionRequest" />
3268 <input> <soap:body use="literal" /> </input>
3269 <output> <soap:body use="literal" /> </output>
3270 </operation>
3271 <operation name="GetObjectInfo">
3272 <soap:operation soapAction="urn:liberty:ps:2006-08:GetObjectInfoRequest" />
3273 <input> <soap:body use="literal" /> </input>
3274 <output> <soap:body use="literal" /> </output>
3275 </operation>
3276 <operation name="SetObjectInfo">
3277 <soap:operation soapAction="urn:liberty:ps:2006-08:SetObjectInfoRequest" />
3278 <input> <soap:body use="literal" /> </input>
3279 <output> <soap:body use="literal" /> </output>
3280 </operation>
3281 <operation name="QueryObjects">
3282 <soap:operation soapAction="urn:liberty:ps:2006-08:QueryObjectsRequest" />
3283 <input> <soap:body use="literal" /> </input>
3284 <output> <soap:body use="literal" /> </output>
3285 </operation>
3286 <operation name="TestMembership">
3287 <soap:operation soapAction="urn:liberty:ps:2006-08:TestMembershipRequest" />
3288 <input> <soap:body use="literal" /> </input>
```

```
3289 <output> <soap:body use="literal" /> </output>
3290 </operation>
3291 <operation name="ResolveIdentifier">
3292 <soap:operation soapAction="urn:liberty:ps:2006-08:ResolveIdentifierRequest" />
3293 <input> <soap:body use="literal" /> </input>
3294 <output> <soap:body use="literal" /> </output>
3295 </operation>
3296
3297 </binding>
3298
3299 <service name="PeopleService">
3300 <port name="PeoplePort" binding="ps:PeopleServiceSoapBinding">
3301
3302 <!-- Modify with the REAL SOAP endpoint -->
3303
3304 <soap:address location="http://example.com/peopleservice" />
3305 </port>
3306 </service>
3307
3308 </definitions>
3309
3310
```

3311 References

3312 Normative

- 3313 [LibertyAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,
3314 Single Sign-On, and Identity Mapping Services Specification," Version v2.0, Liberty Alliance Project (30
3315 July, 2006). <http://www.projectliberty.org/specs>
- 3316 [LibertyDisco] Hodges, Jeff, Cahill, Conor, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0,
3317 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 3318 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
3319 2006). <http://www.projectliberty.org/specs>
- 3320 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
3321 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>
- 3322 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version v2.0, Liberty
3323 Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 3324 [LibertySUBS] Kellomäki, Sampo, eds. "Liberty ID-WSF Subscriptions and Notifications," Version 1.0, Liberty
3325 Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 3326 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
3327 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 3328 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
3329 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
3330 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-
3331 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 3332 [SAMLBind2] Cantor, Scott, Hirsch, Frederick, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March
3333 2005). "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
3334 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf](http://docs.oasis-
3335 open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf)
- 3336 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
3337 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
3338 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards
3339 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- 3340 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.
3341 World Wide Web Consortium W3C Recommendation (9 May 2006). [http://www.w3.org/TR/2006/REC-ws-
addr-core-20060509/](http://www.w3.org/TR/2006/REC-ws-
3342 addr-core-20060509/)
- 3343 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
3344 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
3345 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 3346 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
3347 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
3348 <http://www.w3.org/TR/xmlschema-1/>