



Liberty ID-WSF People Service Specification

Version:

1.0-errata-v1.0

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-1.0-diff-v1.0.pdf

1 **Notice**

2 This document has been prepared by Sponsors of the Liberty Alliance. Permission is hereby granted to use the document
3 solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works of this
4 Specification. Entities seeking permission to reproduce portions of this document for other uses must contact the Liberty
5 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 rights,
7 including without limitation, patent rights. The Sponsors of and any other contributors to the Specification are not and
8 shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual
9 property rights. **This Specification is provided "AS IS", and no participant in the Liberty Alliance makes any**
10 **warranty of any kind, express or implied, including any implied warranties of merchantability, non-infringe-**
11 **ment of third party intellectual property rights, and fitness for a particular purpose.** Implementers of this
12 Specification are advised to review the Liberty Alliance Project's website (<http://www.projectliberty.org/>) for infor-
13 mation concerning any Necessary Claims Disclosure Notices that have been received by the Liberty Alliance
14 Management Board.

15 Copyright © 2007 2FA Technology; Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.; Amer-
16 ican Express Company; Amsoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Axalto; Bank of
17 America Corporation; Beta Systems Software AG; BIPAC; British Telecommunications plc; Computer Associates
18 International, Inc.; Credentica; DataPower Technology, Inc.; Deutsche Telekom AG, T-Com; Diamelle Technologies,
19 Inc.; Diversinet Corp.; Drummond Group Inc.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Falkin Systems
20 LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le développement
21 de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens Ltd.; GSA Office of Governmentwide Policy;
22 Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke & Devrient GmbH; Hewlett-Packard GSA Office
23 Company; Hochhauser & Co., Policy; Hewlett-Packard LLC; IBM Corporation; Intel Corporation; Intuit Inc.; Kant-
24 ega; Kayak Interactive; Livo Technologies; Luminance Consulting Services; MasterCard International; MedCommons
25 Inc.; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; NTT DoCoMo, Inc.; Netegrity, Inc.; Neustar, Inc.;
26 New Zealand Government State Services Commission; Nippon Telegraph and Telephone Corporation; Nokia Corpo-
27 ration; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation; RSA Security
28 Inc.; Reactivity Inc.; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp Laboratories of America;
29 Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.;
30 Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security; Trusted Network Technologies;
31 UNINETT AS; UTI; VeriSign, Inc.; Vodafone Group Plc.; Wave Systems Corp. All rights reserved.

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

38 Contents

39	1. Introduction.....	7
40	1.1. Overview.....	7
41	1.2. Notation.....	7
42	1.3. Terminology.....	8
43	1.4. Namespaces.....	8
44	2. Data Model.....	9
45	2.1. <Object> Element.....	9
46	2.1.1. NodeType Attribute.....	9
47	2.1.2. CreatedDateTime Attribute.....	9
48	2.1.3. ModifiedDateTime Attribute.....	10
49	2.1.4. <ObjectID> Element.....	10
50	2.1.5. <DisplayName> Element.....	10
51	2.1.6. <Tag> Element.....	10
52	2.1.7. <ObjectRef> Element.....	11
53	2.2. <Token> Element.....	11
54	3. People Service.....	12
55	3.1. Overview.....	12
56	3.2. Service Type.....	12
57	3.3. Action URIs.....	12
58	3.4. Request and Response Abstract Types.....	13
59	3.4.1. Complex Type RequestAbstractType.....	13
60	3.4.2. Complex Type ResponseAbstractType.....	13
61	3.5. Status.....	14
62	3.6. Identity Token Policy.....	15
63	3.7. Success & Failure.....	15
64	3.8. Subscription and Notification.....	15
65	3.8.1. <Subscription> Element.....	16
66	3.8.2. Notify and NotifyResponse Messages.....	17
67	3.8.3. <Notification> Element.....	17
68	3.9. Adding an Entity.....	18
69	3.9.1. wsa:Action Values.....	18
70	3.9.2. AddEntityRequest Message.....	18
71	3.9.3. AddEntityResponse Message.....	19
72	3.9.4. Processing Rules.....	20
73	3.10. Adding a Known Entity.....	22
74	3.10.1. wsa:Action Values.....	22
75	3.10.2. AddKnownEntityRequest Message.....	22
76	3.10.3. AddKnownEntityResponse Message.....	23
77	3.10.4. Processing Rules.....	24
78	3.11. Removing an Entity.....	26
79	3.11.1. wsa:Action Values.....	26
80	3.11.2. RemoveEntityRequest Message.....	26
81	3.11.3. RemoveEntityResponse Message.....	26
82	3.11.4. Processing Rules.....	27
83	3.12. Adding a Collection.....	27
84	3.12.1. wsa:Action Values.....	27
85	3.12.2. AddCollectionRequest Message.....	27
86	3.12.3. AddCollectionResponse Message.....	28
87	3.12.4. Processing Rules.....	29
88	3.13. Removing a Collection.....	29
89	3.13.1. wsa:Action Values.....	29

90	3.13.2. RemoveCollectionRequest Message	29
91	3.13.3. RemoveCollectionResponse Message	30
92	3.13.4. Processing Rules.....	30
93	3.14. Adding to a Collection.....	31
94	3.14.1. wsa:Action Values	31
95	3.14.2. AddToCollectionRequest Message	31
96	3.14.3. AddToCollectionResponse Message	32
97	3.14.4. Processing Rules.....	32
98	3.15. Removing from a Collection.....	33
99	3.15.1. wsa:Action Values	33
100	3.15.2. RemoveFromCollectionRequest Message	33
101	3.15.3. RemoveFromCollectionResponse Message	34
102	3.15.4. Processing Rules.....	34
103	3.16. Listing Members	34
104	3.16.1. wsa:Action Values	35
105	3.16.2. ListMembersRequest Message	35
106	3.16.3. ListMembersResponse Message	36
107	3.16.4. Examples	37
108	3.16.5. Processing Rules.....	39
109	3.17. Retrieving Info.....	40
110	3.17.1. wsa:Action Values	40
111	3.17.2. GetObjectInfoRequest Message	40
112	3.17.3. GetObjectInfoResponse Message	41
113	3.17.4. Processing Rules.....	42
114	3.18. Updating Info	42
115	3.18.1. wsa:Action Values	42
116	3.18.2. SetObjectInfoRequest Message.....	42
117	3.18.3. SetObjectInfoResponse Message.....	43
118	3.18.4. Processing Rules.....	43
119	3.19. Querying Objects	44
120	3.19.1. wsa:Action Values	44
121	3.19.2. QueryObjectsRequest Message.....	44
122	3.19.3. QueryObjectsResponse Message.....	45
123	3.19.4. Processing Rules.....	46
124	3.20. Testing Membership	47
125	3.20.1. wsa:Action Values	47
126	3.20.2. TestMembershipRequest Message.....	47
127	3.20.3. TestMembershipResponse Message.....	48
128	3.20.4. Processing Rules.....	48
129	3.21. Resolving Objects	49
130	3.21.1. wsa:Action Values	49
131	3.21.2. ResolveIdentifierRequest Message	49
132	3.21.3. ResolveIdentifierResponse Message	50
133	3.21.4. Processing Rules.....	51
134	4. Interaction with Users.....	53
135	4.1. Model (Informative).....	53
136	4.2. Additional Federations for Sharing of Identity Services	53
137	4.3. Consent Model	54
138	4.4. Elements Supporting Invitation.....	54
139	4.4.1. PStoSPRedirectURL element.....	54
140	4.4.2. <SPToPSRedirectURL> element.....	54
141	4.4.3. <QueryString> element	55
142	5. Sequence Examples	58

143	5.1. Policy definition	58
144	5.2. AccessControl	59
145	5.3. Group Operation	61
146	6. Security Considerations	64
147	7. XML Schema for ID-WSF People Service.....	65
148	8. Abstract WSDL.....	74
149	References.....	80

150 1. Introduction

151 1.1. Overview

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

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

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

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

170 There are two significant implications of this model:

- 171 1. Both users must maintain or establish accounts at the same provider. Typically, the result of this requirement is
172 that the friend being invited to interact (e.g., View vacation photos, etc) is forced to create an account (with
173 associated logins and passwords to remember) at a provider where they might not otherwise choose to do so.
- 174 2. If some connection between two friends is established in the context of the photo site, it can't be leveraged in some
175 other context (e.g., Calendar sharing) unless that provider happens to host both services.

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

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

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

186 1.2. Notation

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

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

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

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

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

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

200 1.3. Terminology

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

206 1.4. Namespaces

207 The following namespaces are used in the schema definitions:

- 208 • The prefix `ps:` stands for the Liberty ID-WSF People Service schema namespace (`urn:liberty:ps:`
209 `2006-08`). This namespace is the default for instance fragments, type names, and element names in this document.
- 210 • The prefix `xs:` stands for the W3C XML schema namespace (`http://www.w3.org/2001/XMLSchema`) [[Sche-](#)
211 [mal-2](#)].
- 212 • The prefix `xml:` stands for the W3C XML namespace (`http://www.w3.org/XML/1998/namespace`)
213 [[XML](#)].
- 214 • The prefix `saml:` stands for the OASIS SSTC SAML2.0 Assertion namespace (`urn:oasis:names:tc:SAML:`
215 `2.0:assertion`) [[SAMLCore2](#)].
- 216 • The prefix `samlp:` stands for the OASIS SSTC SAML2.0 Protocol namespace (`urn:oasis:names:tc:SAML:`
217 `2.0:protocol`) [[SAMLCore2](#)].
- 218 • The prefix `ims:` stands for the Liberty ID-WSF Authentication Service Identity Mapping Service namespace
219 (`urn:liberty:ims:2006-08`) [[LibertyAuthn](#)].
- 220 • The prefix `sec:` stands for the Liberty ID-WSF Security Mechanisms Core namespace (`urn:liberty:sec:`
221 `2005-11`) [[LibertySecMech](#)].
- 222 • The prefix `subs:` stands for the Liberty ID-WSF Subscriptions & Notifications namespace (`urn:liberty:`
223 `ssos:2006-08`) [[LibertySUBS](#)].

224 2. Data Model

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

228 2.1. <Object> Element

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

232 The <Object> element has <DisplayName> elements to carry a human-readable name for the <Object> (see [Section 2.1.5](#)).
233

234 The value of the <ObjectID> element uniquely identifies the <Object> within the set of all <Object> elements that
235 are accessible to a particular consumer of the People Service for the targeted identity.

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

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

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

```
241
242 <xs:element name="Object" type="ObjectType"/>
243 <xs:complexType name="ObjectType">
244   <xs:sequence>
245     <xs:element ref="ObjectID" minOccurs="0"/>
246     <xs:element name="DisplayName" type="LocalizedDisplayNameType"
247       minOccurs="1" maxOccurs="unbounded"/>
248     <xs:element name="Tag" type="TagType" minOccurs="0" maxOccurs="unbounded"/>
249     <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
250     <xs:element name="ObjectRef" type="ObjectIDType" minOccurs="0" maxOccurs="unbounded"/>
251   </xs:sequence>
252   <xs:attribute name="NodeType" type="xs:anyURI" use="required"/>
253   <xs:attribute name="CreatedDateTime" type="xs:dateTime" use="optional"/>
254   <xs:attribute name="ModifiedDateTime" type="xs:dateTime" use="optional"/>
255 </xs:complexType>
256
```

257 2.1.1. NodeType Attribute

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

260 `urn:liberty:ps:collection` If an <Object> has this URI for the value of the `NodeType` attribute, it represents a collection
261 that has zero or more <Object> as child elements. The child <Object> elements may have
262 a `NodeType` of either `urn:liberty:ps:collection` or `urn:liberty:ps:entity`.

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

266 2.1.2. CreatedDateTime Attribute

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

268 **2.1.3. ModifiedDateTime Attribute**

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

271 **2.1.4. <ObjectID> Element**

272 The <ObjectID> element is defined so that WSCs the PS provider can refer, unambiguously, to the parent
273 <Object> and elements.

```
274
275     <!-- Declaration of ObjectID element -->
276     <xs:element name="ObjectID" type="ObjectIDType"/>
277     <!-- Definition of ObjectIDType -->
278     <xs:complexType name="ObjectIDType">
279         <xs:simpleContent>
280             <xs:restriction base="xs:anyURI" />
281         </xs:simpleContent>
282     </xs:complexType>
283
```

284 The PS provider controls the creation of object identifiers. When a WSC requests the creation of an object, the WSC
285 MUST NOT provide an ObjectID in such a request message. If the request is successful, the PS provider MUST return
286 an object identifier for the new object in an ObjectID element in its response message - the WSC MUST use this
287 returned identifier in subsequent operations on that object.

288 Where privacy is a concern, PS providers MUST ensure that ObjectIDs do not create a privacy concern by allowing
289 different WSCs to make inappropriate correlations about the users for which the Object identifiers stand. Unique
290 identifiers for different WSCs (e.g., pairwise identifiers), reuse of identifiers across identifiers different services, and
291 encrypted identifiers are potential mechanisms for addressing this concern.

292 **2.1.5. <DisplayName> Element**

293 The <DisplayName> element provides a human-readable friendly name for Objects. The value of this element
294 SHOULD NOT be used to uniquely identify Objects; rather the ObjectID element SHOULD be used. (see Sec-
295 tion 2.1.4).

```
296
297 <xs:complexType name="LocalizedDisplayNameType">
298     <xs:simpleContent>
299         <xs:extension base="xs:string">
300             <xs:attribute name="Locale" type="xs:language" use="optional"/>
301             <xs:attribute name="IsDefault" type="boolean" use="optional"/>
302         </xs:extension>
303     </xs:simpleContent>
304 </xs:complexType>
305
```

306 The <Locale> attribute specifies the language in which the display name is expressed. If not present, providers
307 SHOULD determine how to best display the name through other means.

308 The <IsDefault> attribute identifies which <DisplayName> element, if there are multiple, is default. There MUST
309 NOT be more than one <DisplayName> element with IsDefault set as "true."

310 **2.1.6. <Tag> Element**

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

```
314
315 <xs:complexType name="TagType">
316   <xs:attribute name="Ref" type="xs:anyURI" use="required"/>
317 </xs:complexType>
318
```

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

321 If they understand the tag space for a <Tag> element, WSCs and PS providers MAY process as appropriate. WSCs
322 and PS providers MAY ignore <Tag> elements.

323 **2.1.7. <ObjectRef> Element**

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

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

328 **2.2. <Token> Element**

329 The <sec:Token> element acts as a container for identity tokens, see [LibertySecMech]

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

332 **3. People Service**

333 **3.1. Overview**

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

339 **3.2. Service Type**

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

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

342 **3.3. Action URIs**

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

345 This specification defines the following action identifiers:

- 346 • *urn:liberty:ps:2006-08:AddEntityRequest*
- 347 • *urn:liberty:ps:2006-08:AddEntityResponse*
- 348 • *urn:liberty:ps:2006-08:AddKnownEntityRequest*
- 349 • *urn:liberty:ps:2006-08:AddKnownEntityResponse*
- 350 • *urn:liberty:ps:2006-08:RemoveEntityRequest*
- 351 • *urn:liberty:ps:2006-08:RemoveEntityResponse*
- 352 • *urn:liberty:ps:2006-08:AddCollectionRequest*
- 353 • *urn:liberty:ps:2006-08:AddCollectionResponse*
- 354 • *urn:liberty:ps:2006-08:RemoveCollectionRequest*
- 355 • *urn:liberty:ps:2006-08:RemoveCollectionResponse*
- 356 • *urn:liberty:ps:2006-08:AddToCollectionRequest*
- 357 • *urn:liberty:ps:2006-08:AddToCollectionResponse*
- 358 • *urn:liberty:ps:2006-08:RemoveFromCollectionRequest*
- 359 • *urn:liberty:ps:2006-08:RemoveFromCollectionResponse*
- 360 • *urn:liberty:ps:2006-08:ListMembersRequest*
- 361 • *urn:liberty:ps:2006-08:ListMembersResponse*
- 362 • *urn:liberty:ps:2006-08:GetObjectInfoRequest*

- 363 • *urn:liberty:ps:2006-08:GetObjectInfoResponse*
- 364 • *urn:liberty:ps:2006-08:SetObjectInfoRequest*
- 365 • *urn:liberty:ps:2006-08:SetObjectInfoResponse*
- 366 • *urn:liberty:ps:2006-08:QueryObjectsRequest*
- 367 • *urn:liberty:ps:2006-08:QueryObjectsResponse*
- 368 • *urn:liberty:ps:2006-08:TestMembershipRequest*
- 369 • *urn:liberty:ps:2006-08:TestMembershipResponse*
- 370 • *urn:liberty:ps:2006-08:ResolveIdentifierRequest*
- 371 • *urn:liberty:ps:2006-08:ResolveIdentifierResponse*
- 372 • *urn:liberty:ps:2006-08:Notify*
- 373 • *urn:liberty:ps:2006-08:NotifyResponse*

374 3.4. Request and Response Abstract Types

375 3.4.1. Complex Type RequestAbstractType

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

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

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

```
379
380 <!-- Definition of RequestAbstractType -->
381 <xs:complexType name="RequestAbstractType" abstract="true">
382   <xs:anyAttribute namespace="##other" processContents="lax"/>
383 </xs:complexType>
384
```

385 3.4.2. Complex Type ResponseAbstractType

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

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

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

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

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

```
393
394 <!-- Definition of ResponseAbstractType -->
395 <xs:complexType name="ResponseAbstractType" abstract="true">
396   <xs:sequence>
397     <xs:element ref="lu:Status"/>

```

```
398     </xs:sequence>
399     <xs:anyAttribute namespace="##other" processContents="lax" />
400 </xs:complexType>
401
```

402 3.5. Status

403 All the response messages extended from `ResponseAbstractType` contain a `<lu:Status>` element (see [Section 3.4.2](#)) to indicate whether or not the processing of the request message has succeeded. The `<lu:Status>` element
404 is included from the Liberty ID-WSF Utility Schema. A `<lu:Status>` element MAY contain other `<lu:Status>`
405 elements providing more detailed information. A `<lu:Status>` element has a `code` attribute, which contains the return
406 status as a string. The local definition of these codes is specified in this document. This specification defines the
407 following status codes to be used as values for the `code` attribute:
408

- 409 • `CannotFindIDP`
- 410 • `CannotFindObject`
- 411 • `CannotResolveToken`
- 412 • `CircularCollection`
- 413 • `DuplicateObject`
- 414 • `Failed`
- 415 • `InvalidNodeType`
- 416 • `InvalidObjectID`
- 417 • `ObjectIsCollection`
- 418 • `ObjectIsEntity`
- 419 • `OK`
- 420 • `OKButNoSubscription`
- 421 • `NoResults`
- 422 • `NoSubscribeWithOffset`
- 423 • `NoTargetSpecified`
- 424 • `PartialSuccess`
- 425 • `PolicyDoesNotAllow`
- 426 • `ResolveIdentifierNotSupported`
- 427 • `SubscribeToChildrenOnly`
- 428 • `Timeout`
- 429 • `UnexpectedError`
- 430 • `UnrecognizedFilter`

431 • UnrecognizedNamespace

432 • UnspecifiedError

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

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

440 *OKButNoSubscription* The value *OKButNoSubscription* means that the processing of the primary request message
441 has succeeded but a *Subscription* request within that message has been rejected. A second
442 level status code MAY be used to indicate some special cases.

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

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

448 3.6. Identity Token Policy

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

454 The <sec:TokenPolicy> element serves as a container for such WSC policy requirements. The <sec:
455 TokenPolicy> element is defined in [LibertySecMech]

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

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

460 3.7. Success & Failure

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

464 3.8. Subscription and Notification

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

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

470 The subscription & notification model used within this specification can be considered a constrained version of the
471 more flexible model defined in [LibertySUBS].

472 3.8.1. <Subscription> Element

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

476 The schema declaration for the <Subscription> element is derived from the correspondingly named type defined
477 in [LibertySUBS]. The schema declaration is shown below:

```
478  
479     <xs:element name="Subscription" type="subs:SubscriptionType"/>  
480
```

481 3.8.1.1. Selecting Objects

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

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

486 3.8.1.2. Triggers

487 This specification defines no triggers.

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

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

493 3.8.1.3. Subscription Start

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

495 For a <Subscription> sent within an <AddEntityRequest> message, the *starts* attribute SHOULD be omitted.
496 If present, the PS provider MAY ignore the attribute.

497 3.8.1.4. Subscription Aggregation

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

499 There MUST be no <Aggregation> element present in a subscription.

500 3.8.1.5. Subscription Expiration

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

502 Unless the value of the *expires* attribute specifies that expiration should occur earlier, for a <Subscription> sent
503 within an <AddEntityRequest> message, expiration is considered to have occurred at such time as the PS provider
504 has delivered a <Token> for the invited user to the WSC and the WSC has acknowledged its receipt.

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

507 3.8.1.6. Subscription Querying and Management

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

510 3.8.1.7. Including Data

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

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

514 For a <Subscription> sent within an <AddEntityRequest> message, the *includeData* attribute SHOULD be
515 omitted.

516 3.8.2. Notify and NotifyResponse Messages

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

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

521 The schema declarations for the <Notify> and <NotifyResponse> messages are derived from the correspondingly
522 named types defined in [LibertySUBS]. The schema declarations are shown below:

```
523 <xs:element name="Notify" type="NotifyType" />
524
525
526 <xs:complexType name="NotifyType">
527   <xs:complexContent>
528     <xs:extension base="RequestAbstractType">
529       <xs:sequence>
530         <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />
531       </xs:sequence>
532       <xs:attributeGroup ref="subs:NotifyAttributeGroup" />
533     </xs:extension>
534   </xs:complexContent>
535 </xs:complexType>
536
537 <xs:element name="NotifyResponse" type="subs:NotifyResponseType" />
538
```

539 3.8.3. <Notification> Element

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

542 The value of the SubscriptionID attribute on the <Notification> element MUST match that of the
543 SubscriptionID attribute on the <Subscription> element corresponding to which the <Notification> is being
544 sent.

545 The schema declaration for the <Notification> element is derived from the correspondingly named type defined
546 in [LibertySUBS]. The schema declaration is shown below:

```
547 <xs:element name="Notification" type="NotificationType" />
548
549
550 <xs:complexType name="NotificationType">
551   <xs:complexContent>
```

```

552         <xs:extension base="subs:NotificationType">
553             <xs:sequence>
554                 <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded"/>
555             </xs:sequence>
556         </xs:extension>
557     </xs:complexContent>
558 </xs:complexType>
559

```

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

```

561
562     <xs:element name="ItemData" type="ItemDataType"/>
563
564     <xs:complexType name="ItemDataType">
565         <xs:choice>
566             <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
567             <xs:element ref="sec:Token" minOccurs="0"/>
568         </xs:choice>
569     </xs:complexType>
570

```

571 3.9. Adding an Entity

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

574 The Object element created by an <AddEntityRequest> message becomes a direct child of the root node.

575 3.9.1. wsa:Action Values

576 <AddEntityRequest> messages MUST include a <wsa:Action> SOAP header with the value of "**urn:liberty:ps:
577 2006-08:AddEntityRequest.**" <AddEntityResponse> messages MUST include a <wsa:Action> SOAP header
578 with the value of "**urn:liberty:ps:2006-08:AddEntityResponse.**"

579 3.9.2. AddEntityRequest Message

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

581 The <AddEntityRequest> MUST NOT be used to add a new Object to an existing Object, nor to create two nested
582 Objects.

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

587 A PS provider can also itself use the <AddEntityRequest> message to request that an Object be added to a PS list.
588 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
589 added to the user's PS.

590 The <AddEntityRequest> message has the complex type **AddEntityRequestType**, which extends
591 **RequestAbstractType** and adds the following elements:

592 <Object> [**Re-** The <Object> element is used to convey the target user Object being added.
593 **quired**]

594 <PStoSPRedirectU The <PStoSPRedirectURL> element is used to convey the URL to which a PS provider will
595 RL> [**Optional**] redirect the invited users after federating their IDP account to the PS provider.

- 595 <CreatePSObject> The <CreatePSObject> element allows a WSC to indicate that it desires the PS provider
596 **[Optional]** create (or verify the existence of) an `Object` for the inviting user at the PS provider of the
597 invited user (i.e., create or verify the reciprocal relationship).
- 598 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
599 **[Optional]** that the PS provider return to it (when later possible) an identity token for the invited user.
- 600 <sec:TokenPolicy> [Optional] The <sec:TokenPolicy> element is used as a container for the WSC's policy requirements
601 of any identity token that it might ask to be returned.

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

```
603
604 <!-- Declaration of AddEntityRequest element -->
605 <xs:element name="AddEntityRequest" type="AddEntityRequestType"/>
606 <!-- Definition of AddEntityRequestType -->
607 <xs:complexType name="AddEntityRequestType">
608   <xs:complexContent>
609     <xs:extension base="RequestAbstractType">
610       <xs:sequence>
611         <xs:element ref="Object"/>
612         <xs:element ref="PStoSPRedirectURL" minOccurs="0"/>
613         <xs:element ref="CreatePSObject" minOccurs="0"/>
614         <xs:element ref="Subscription" minOccurs="0"/>
615         <xs:element ref="TokenPolicy" minOccurs="0"/>
616       </xs:sequence>
617     </xs:extension>
618   </xs:complexContent>
619 </xs:complexType>
620
```

621 The following is an example of an <AddEntityRequest> message used to create an `Object` for a user. The WSC is
622 not requesting that an identity token for the newly created user `Object` be returned. If it desired this, it would include
623 a <Subscription> element and, optionally, a <sec:TokenPolicy> element indicating its requirements of the re-
624 turned identity token.

```
625
626 <AddEntityRequest>
627   <Object NodeType="urn:liberty:ps:entity">
628     <DisplayName>Alison</DisplayName>
629   </Object>
630   <PStoSPRedirectURL>some SP URL</PStoSPRedirectURL>
631 </AddEntityRequest>
632
```

633 3.9.3. AddEntityResponse Message

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

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

- 638 <Object> **[Optional]** The <Object> element is used to convey the `Object` element just created at the PS provider.
- 639 <SptoPSRedirectURL> **[Optional]** The <SptoPSRedirectURL> element is used to convey the URL to which the PS provider
640 desires the invited user be sent if and when they respond to the invitation that the SP will
641 compose and deliver.

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

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

650 If an issuer of the artifact intends to exchange SAML messages over SAML protocol[[SAML-
 651 Core2](#)], the value of the artifact itself, and optional relay state information conveyed in the
 652 <QueryString> element, MUST satisfy the formatting and encoding requirements of the
 653 SAML Artifact Binding (see [[SAMLBind2](#)]) as specified by the URI identifier *urn:oasis:
 654 names:tc:SAML:2.0:artifact-04*.

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

```
656
657 <!-- Declaration of AddEntityResponse element -->
658 <xs:element name="AddEntityResponse" type="AddEntityResponseType"/>
659 <!-- Definition of AddEntityResponseType -->
660 <xs:complexType name="AddEntityResponseType">
661   <xs:complexContent>
662     <xs:extension base="ResponseAbstractType">
663       <xs:sequence>
664         <xs:element ref="Object" minOccurs="0"/>
665         <xs:element ref="SPtoPSRedirectURL" minOccurs="0"/>
666         <xs:element ref="QueryString" minOccurs="0"/>
667       </xs:sequence>
668     </xs:extension>
669   </xs:complexContent>
670 </xs:complexType>
671
```

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

```
676
677 <AddEntityResponse>
678   <Status code="OK"/>
679   <Object NodeType="urn:liberty:ps:entity">
680     <ObjectID>https://ps.com/kudfhgs</ObjectID>
681     <DisplayName>Alison</DisplayName>
682   </Object>
683   <SPtoPSRedirectURL>some PS URL</SPtoPSRedirectURL>
684 </AddEntityResponse>
685
```

686 3.9.4. Processing Rules

687 The WSC:

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

- 690 • MUST include at least one <DisplayName> element for the invited user within the <Object> element. This
691 element contains the friendly name that the user desires be used for the created *urn:liberty:ps:entity* Object.
- 692 • MAY include a <PStoSPRedirectURL> element.
- 693 • MAY, if it desires that the PS provider create (if not already existing) an Object for the inviting user at the PS
694 provider of the invited user, include a <CreatePSObject> element.
- 695 • MAY, if it desires that an identity token be returned to it through a subsequent <Notification> message, include
696 a <Subscription> element. The presence of a <Subscription> element indicates to the PS provider that the
697 SP desires that the PS provider return to it (when later possible) an identity token for the invited user within a
698 <Notification> in a <Notify> message - this is possible after a federation has been established between the
699 PS provider and the appropriate IDP. If no <Subscription> element is present, the SP is indicating that the PS
700 provider need not return an identity token through this mechanism.
- 701 If the WSC includes a <Subscription> element, it MAY specify its requirements of the eventually returned
702 identity token by including a <sec:TokenPolicy> element. The WSC SHOULD NOT include a <sec:
703 TokenPolicy> element unless also including a <Subscription> element.
- 704 In responding to a successful ~~an~~ <AddEntityRequest> message, the PS provider:
- 705 • MUST create a new Object element as a direct child of the root node.
- 706 • MUST include an ObjectID in the returned Object.
- 707 • SHOULD include either or both of a <SPToPSRedirectURL> or <QueryString> element in the
708 <AddEntityResponse> message returned to the calling SP.
- 709 • MUST be prepared for the invited user to, at some point in the future, visit the URL provided in any specified
710 <SPToPSRedirectURL> element. As it may be some time before the invited user does respond, the PS provider
711 SHOULD store such a URL for a reasonable length of time.
- 712 MUST, if and when the invited user does respond to the URL specified by the <SPToPSRedirectURL> element,
713 endeavor to establish a federated identifier for that user with the appropriate identity provider (see [Section 4](#))
- 714 SHOULD, if and when such a federated identifier is established, send an <ims:IdentityMappingRequest>
715 message to that IDP requesting a long-lived identity token (targeted at itself as the provider) for the user for which
716 the federated identifier was just established.
- 717 • SHOULD, if the <AddEntityRequest> message contained a <CreatePSObject> element, **attempt to create or**
718 **verify the existence of** ~~be~~ an object for the inviting user in the PS of the invited user **(when made possible by a**
719 **federated identifier being established for the** ~~user-~~ **invited user)**.
- 720 It may be the case that the inviting user is **already** in the PS of the invited user as a result of a prior invitation
721 sequence initiated "**from the other side.**" The PS of the inviting user MUST ensure that no duplicate object be added.
- 722 **MAY**, in order to determine whether an object for the inviting user already exists, query the members of the PS of
723 the invited user using the <ListMembersRequest> message and ask the invited user to assist in determining
724 whether the inviting user is already in the list. Other mechanisms (e.g., using a ~~message-~~
725 <TestMembershipRequest>) for making this determination MAY alternatively be used.
- 726 SHOULD, if there is no existing object for the inviting user, request that an object be created with either the
727 <AddEntityRequest> or <AddKnownEntityRequest> messages.

728 SHOULD, if sending an `<AddKnownEntityRequest>` message for the addition, include a `<sec:Token>` element
729 carrying a token for the inviting user - this `<sec:Token>` obtained from the Identity Mapping Service of ~~user~~ the
730 inviting user.

731 • SHOULD, if the `<AddEntityRequest>` message contained a `<Subscription>` element, send an `<ims:`
732 `IdentityMappingRequest>` message to that IDP requesting an identity token for the user for which the federated
733 identifier was established but in the namespace of the requesting SP.

734 This `<ims:IdentityMappingRequest>` message to the IDP MUST include any policy directives present in the
735 `<AddEntityRequest>`.

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

740 3.10. Adding a Known Entity

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

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

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

753 3.10.1. wsa:Action Values

754 `<AddKnownEntityRequest>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:lib-`
755 `erty:ps:2006-08:AddKnownEntityRequest.`" `<AddKnownEntityResponse>` messages MUST include a `<wsa:`
756 `Action>` SOAP header with the value of "`urn:liberty:ps:2006-08:AddKnownEntityResponse.`"

757 3.10.2. AddKnownEntityRequest Message

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

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

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

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

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

- 767 <sec:Token> [**Re-** The <sec:Token> element is used to convey an identity token for the target user Object
768 **quired]** being created.
- 769 <CreatePSObject> The <CreatePSObject> element is used as a directive with which a WSC indicates that it
770 [**Optional]** desires a PS provider create (or verify the existence of) an Object for the inviting user at the
771 PS provider of the invited user.
- 772 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
773 [**Optional]** that the PS provider return to it (when later possible) an identity token for the invited user.
- 774 <sec: The <sec:TokenPolicy> element is used as a container for a WSC's requirements to an
775 TokenPolicy> [**Op- identity token.**
776 **tional]**

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

```
777
778 <!-- Declaration of AddKnownEntityRequest element -->
779 <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType"/>
780 <!-- Definition of AddKnownEntityRequestType -->
781 <xs:complexType name="AddKnownEntityRequestType">
782   <xs:complexContent>
783     <xs:extension base="RequestAbstractType">
784       <xs:sequence>
785         <xs:element ref="Object"/>
786         <xs:element ref="sec:Token"/>
787         <xs:element ref="CreatePSObject" minOccurs="0"/>
788         <xs:element ref="Subscription" minOccurs="0"/>
789         <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
790       </xs:sequence>
791     </xs:extension>
792   </xs:complexContent>
793 </xs:complexType>
794
```

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

```
796
797 <AddKnownEntityRequest>
798   <Object NodeType="urn:liberty:ps:entity">
799     <DisplayName>Bob</DisplayName>
800   </Object>
801   <sec:Token>
802     <saml:Assertion>
803       <saml:Subject>
804         <saml:NameID></saml:NameID>
805       </saml:Subject>
806     </saml:Assertion>
807   </sec:Token>
808 </AddKnownEntityRequest>
809
```

810 3.10.3. AddKnownEntityResponse Message

811 A PS provider responds to an <AddKnownEntityRequest> message with an <AddKnownEntityResponse> mes-
812 sage, in which the PS provider MAY contain the newly created <Object> element.

813 The <AddKnownEntityResponse> message has the complex type **AddKnownEntityResponseType**, which extends
814 **ResponseAbstractType** and adds the following elements:

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

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

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

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

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

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

```
833
834 <!-- Declaration of AddKnownEntityResponse element -->
835 <xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType"/>
836 <!-- Definition of AddKnownEntityResponseType -->
837 <xs:complexType name="AddKnownEntityResponseType">
838   <xs:complexContent>
839     <xs:extension base="ResponseAbstractType">
840       <xs:sequence>
841         <xs:element ref="Object" minOccurs="0"/>
842         <xs:element ref="SptoPSRedirectURL" minOccurs="0" maxOccurs="1"/>
843         <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
844       </xs:sequence>
845     </xs:extension>
846   </xs:complexContent>
847 </xs:complexType>
848
```

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

```
852
853 <AddKnownEntityResponse>
854   <Status code="OK"/>
855   <Object NodeType="urn:liberty:ps:entity">
856     <ObjectID>https://ps.com/lafnervf</ObjectID>
857     <DisplayName>Bob</DisplayName>
858   </Object>
859 </AddKnownEntityResponse>
860
```

861 3.10.4. Processing Rules

862 The WSC:

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

- 865 • MUST include a <Token> element within the <AddKnownEntityRequest> message.
- 866 When the token is not an identity token (as is the likely case when the known identifier is provided by the inviting
867 user), the WSC SHOULD use a SAML <saml:NameID> element within the <Token> element. If the WSC knows
868 the format of the known identifier, it SHOULD use the appropriate value for the Format attribute on the <saml:
869 NameID> element.
- 870 • MUST, if a SAML <Assertion> is used as the identity token format, specify the known identifier in that assertion's
871 <Subject> element.
- 872 • MUST include at least one <DisplayName> for the invited user within the <Object> element. This element
873 contains the friendly name that the user desires be used for the created user Object.
- 874 • MAY, if it desires that the PS provider create (or verify the existence of) an Object for the inviting user at the PS
875 provider of the invited user, include a <CreatePSObject> element.
- 876 • MAY, if it desires that an identity token be returned to it through a subsequent <Notification> message, include
877 a <Subscription> element. The presence of a <Subscription> element indicates to the PS provider that the
878 SP desires that the PS provider return to it (when later possible) an identity token for the invited user within a
879 <Notification> message - this possible after a federation has been established between the PS provider and the
880 appropriate IDP. If no <Subscription> element is present, the SP is indicating that the PS provider need not
881 return an identity token through this mechanism.
- 882 In responding to an <AddKnownEntityRequest> message, the PS provider:
- 883 • MAY include either or both of a <SPtoPSRedirectURL> or <QueryString> element in the
884 <AddKnownEntityResponse> message returned to the calling SP.
- 885 • SHOULD, if the <AddKnownEntityRequest> message contained a <Subscription> element, send a <ims:
886 IdentityMappingRequest> message to that IDP requesting an identity token for the user for which the federated
887 identifier was established but in the namespace of the requesting SP.
- 888 This <ims:IdentityMappingRequest> message to the IDP MUST include any policy directives present in the
889 <AddKnownEntityRequest>.
- 890 • SHOULD, if the <AddKnownEntityRequest> message contained a <Subscription> element and the <ims:
891 IdentityMappingRequest> message to the IDP resulted in an identity token for the user being returned, forward
892 on this identity token to the SP within a <Notification> message corresponding to the original
893 <Subscription> element.
- 894 • SHOULD, if the <AddKnownEntityRequest> message contained a <CreatePSObject> element, ensure that
895 there be an object for the inviting user in the PS of the invited user.
- 896 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
897 initiated "from the other side." The PS of the inviting user MUST ensure that no duplicate object be added.
- 898 SHOULD, in order to determine whether an object for the inviting user already exists, query the members of the
899 PS of the invited user using the <ListMembersRequest> message.
- 900 SHOULD, if there is no existing object for the inviting user, request that an object be created with either the
901 <AddEntityRequest> or <AddKnownEntityRequest> messages.
- 902 SHOULD, if sending a <AddKnownEntityRequest> message for the addition, include a <sec:Token> element
903 carrying a token for the inviting user.

904 3.11. Removing an Entity

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

908 3.11.1. wsa:Action Values

909 <RemoveEntityRequest> messages MUST include a <wsa:Action> SOAP header with the value of "*urn:liber-*
910 *ty:ps:2006-08:RemoveEntityRequest.*" <RemoveEntityResponse> messages MUST include a <wsa:Action>
911 SOAP header with the value of "*urn:liberty:ps:2006-08:RemoveEntityResponse.*"

912 3.11.2. RemoveEntityRequest Message

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

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

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

921 <TargetObjectID> The <TargetObjectID> element is used to convey one or more ObjectIDs of the target
922 **[Required]** user Objects being removed.

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

```
924
925 <!-- Declaration of RemoveEntityRequest element -->
926 <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType"/>
927 <!-- Definition of RemoveEntityRequestType -->
928 <xs:complexType name="RemoveEntityRequestType">
929   <xs:complexContent>
930     <xs:extension base="RequestAbstractType">
931       <xs:sequence>
932         <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
933       </xs:sequence>
934     </xs:extension>
935   </xs:complexContent>
936 </xs:complexType>
937
```

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

```
939
940 <RemoveEntityRequest>
941   <TargetObjectID>https://ps.com/lafnervf</TargetObjectID>
942 </RemoveEntityRequest>
943
```

944 3.11.3. RemoveEntityResponse Message

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

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

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

```
950  
951 <!-- Declaration of RemoveEntityResponse element -->  
952 <xs:element name="RemoveEntityResponse" type="ResponseAbstractType" />  
953
```

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

```
956  
957 <RemoveEntityResponse>  
958     <Status code="OK" />  
959 </RemoveEntityResponse>  
960
```

961 3.11.4. Processing Rules

962 The WSC:

- 963 • MUST ensure that the targeted <Object> has a Node**Type** attribute with a value of *urn:liberty:ps:entity*.

964 The PS provider:

- 965 • MUST, if the specified user object is not a direct member of the targeted group object, respond with Failed as the
966 code attribute of the top level <lu:Status> element. A second level <lu:Status> MAY be inserted. If so, the
967 code attribute of that second level <lu:Status> element MUST be set with the following status code:

- 968 • NotDirectChild

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

970 3.12. Adding a Collection

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

973 3.12.1. wsa:Action Values

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

977 3.12.2. AddCollectionRequest Message

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

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

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

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

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

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

```

987
988 <!-- Declaration of AddCollectionRequest element -->
989 <xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
990 <!-- Definition of AddCollectionRequestType -->
991 <xs:complexType name="AddCollectionRequestType">
992   <xs:complexContent>
993     <xs:extension base="RequestAbstractType">
994       <xs:sequence>
995         <xs:element ref="Object"/>
996         <xs:element ref="Subscription" minOccurs="0"/>
997       </xs:sequence>
998     </xs:extension>
999   </xs:complexContent>
1000 </xs:complexType>
1001
```

1002 The following is an example of an <AddCollectionRequest> message used to create an Object for a group.

```

1003
1004 <AddCollectionRequest>
1005   <Object NodeType="urn:liberty:ps:collection">
1006     <DisplayName>Soccer Team</DisplayName>
1007   </Object>
1008 </AddCollectionRequest>
1009
```

1010 3.12.3. AddCollectionResponse Message

1011 A PS provider responds to an <AddCollectionRequest> message with an <AddCollectionResponse> message
1012 containing the newly created <Object> element.

1013 The <AddCollectionResponse> message has the complex type **AddCollectionResponseType**, which extends
1014 **ResponseAbstractType** and adds the following element:

1015 <Object> **[Optional]** The <Object> element is used to convey the Object element just created at the PS provider.

1016 The schema declaration for the <AddCollectionResponse> message is shown below.

```

1017
1018 <!-- Declaration of AddCollectionResponse element -->
1019 <xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>
1020 <!-- Definition of AddCollectionResponseType -->
1021 <xs:complexType name="AddCollectionResponseType">
1022   <xs:complexContent>
1023     <xs:extension base="ResponseAbstractType">
1024       <xs:sequence>
1025         <xs:element ref="Object" minOccurs="0"/>
1026       </xs:sequence>
1027     </xs:extension>
1028   </xs:complexContent>
1029 </xs:complexType>
1030
```

1031 The following is an example of an <AddCollectionResponse> to the <AddCollectionRequest> message above.
1032 The PS provider is responding that the request that the Soccer Team be added was successful and returns the created
1033 <Object> element.

```

1034
1035 <AddCollectionResponse>
1036   <Status code="OK"/>
1037   <Object NodeType="urn:liberty:ps:collection">
1038     <ObjectID>https://ps.com/roqlsfof</ObjectID>
```

```
1039         <DisplayName>Soccer Team</DisplayName>
1040     </Object>
1041 </AddCollectionResponse>
1042
```

1043 3.12.4. Processing Rules

1044 The WSC:

- 1045 • MUST include an <Object> element within the <AddCollectionRequest> message.
 - 1046 • MUST include a *NodeType* attribute on the <Object> element with a value of *urn:liberty:ps:collection*.
 - 1047 • MUST include at least one <DisplayName> element for a group within the <Object>. This element contains the
 - 1048 friendly name that the user desires be used for the created group Object.
- 1049 In responding to an <AddCollectionRequest> message, the PS provider:

- 1050 • MUST return an <Object> element within the <AddCollectionResponse> message with the same
- 1051 <DisplayName> as specified on the <AddCollectionRequest>.
- 1052 • MUST include an <ObjectID> element for the newly created group Object.

1053 3.13. Removing a Collection

1054 A WSC indicates to the PS provider that it wishes a group Object to be removed by sending a

1055 <RemoveCollectionRequest> message. The Object being removed MUST be a *urn:liberty:ps:collection*

1056 Object.

1057 3.13.1. wsa:Action Values

1058 <RemoveCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of "**urn:**

1059 **liberty:ps:2006-08:RemoveCollectionRequest.**" <RemoveCollectionResponse> messages MUST include a

1060 <wsa:Action> SOAP header with the value of "**urn:liberty:ps:2006-08:RemoveCollectionResponse.**"

1061 3.13.2. RemoveCollectionRequest Message

1062 A WSC uses the <RemoveCollectionRequest> message to request that a group Object corresponding to the value

1063 of the specified <TargetObjectID> be removed.

1064 The <RemoveCollectionRequest> message is used to completely remove a group Object from the PS resource.

1065 To simply remove a child group <Object> element from some parent group <Object>, the

1066 <RemoveCollectionRequest> message MUST NOT be used, but rather a <RemoveFromCollectionRequest>

1067 with the parent Object's ObjectID specified in the TargetObjectID element, MUST be used (see [Section 3.15](#) for

1068 more details).

1069 The <RemoveCollectionRequest> message does not result in the removal of any child <Object> elements unless

1070 they are explicitly identified through separate <TargetObjectID> elements.

1071 The <RemoveCollectionRequest> message has the complex type **RemoveCollectionRequestType**, which ex-

1072 tends **RequestAbstractType** and adds the following element:

1073 <TargetObjectID> The <TargetObjectID> element specifies the ObjectID of the targeted group Objects

1074 **[Required]** being removed.

1075 The schema declaration for the <RemoveCollectionRequest> message is shown below.

```

1076
1077 <!-- Declaration of RemoveCollectionRequest element -->
1078 <xs:element name="RemoveCollectionRequest" type="RemoveCollectionRequestType"/>
1079 <!-- Definition of RemoveCollectionRequestType -->
1080 <xs:complexType name="RemoveCollectionRequestType">
1081   <xs:complexContent>
1082     <xs:extension base="RequestAbstractType">
1083       <xs:sequence>
1084         <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
1085       </xs:sequence>
1086     </xs:extension>
1087   </xs:complexContent>
1088 </xs:complexType>
1089

```

1090 The following is an example of a <RemoveCollectionRequest> message used to remove an Object for a group.

```

1091
1092 <RemoveCollectionRequest>
1093   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
1094 </RemoveCollectionRequest>
1095

```

1096 3.13.3. RemoveCollectionResponse Message

1097 A PS provider responds to a <RemoveCollectionRequest> message with a <RemoveCollectionResponse>
1098 element. A PS provider removes the specified <Object> and responds a status of this process based on the processing
1099 rules described in [Section 3.11.4](#).

1100 The <RemoveCollectionResponse> message has the type of **ResponseAbstractType**

1101 The schema declaration for the <RemoveCollectionResponse> message is shown below.

```

1102
1103 <!-- Declaration of RemoveCollectionResponse element -->
1104 <xs:element name="RemoveCollectionResponse" type="ResponseAbstractType"/>
1105

```

1106 The following is an example of a <RemoveCollectionResponse> to the <RemoveCollectionRequest> message
1107 above. The PS provider is responding that the request that a specified Object be removed was successful.

```

1108
1109 <RemoveCollectionResponse>
1110   <Status code="OK"/>
1111 </RemoveCollectionResponse>
1112

```

1113 3.13.4. Processing Rules

1114 The WSC:

- 1115 • MUST include a `NodeType` attribute on the <Object> element with a value of *urn:liberty:ps:collection*.

1116 The PS provider:

- 1117 • MUST remove the specified <Object> from the PS list.
- 1118 • MUST NOT, if the specified group <Object> has one or more child <Object> elements, remove any the child
1119 Objects unless those child <Object>s are explicitly specified by their own <ObjectID> values in separate
1120 <TargetObjectID> elements.

1121 3.14. Adding to a Collection

1122 A WSC uses the <AddToCollectionRequest> message to request that child Object elements be added to an existing
1123 group Object. Both user and group Objects can be added to a parent group Object with the
1124 <AddToCollectionRequest> message.

1125 3.14.1. wsa:Action Values

1126 <AddToCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of "urn:
1127 liberty:ps:2006-08:AddToCollectionRequest." <AddToCollectionResponse> messages MUST include a <wsa:
1128 Action> SOAP header with the value of "urn:liberty:ps:2006-08:AddToCollectionResponse."

1129 3.14.2. AddToCollectionRequest Message

1130 The target parent group Object to which child Objects are to be added is indicated by the value of the
1131 <TargetObjectID> element within the <AddToCollectionRequest> element. The child Objects being added
1132 are specified by the values of the <ObjectID> elements within the <AddToCollectionRequest> element.

1133 The <AddToCollectionRequest> message has the complex type **AddToCollectionRequestType**, which extends
1134 **RequestAbstractType** and adds the following elements:

1135 <TargetObjectID> The <TargetObjectID> element is used to convey the ObjectID of the target group
1136 **[Required]** Object to which specified Objects are added.

1137 <ObjectID> **[Re-** The <ObjectID> element is used to convey ObjectIDs of the Objects to be added to the
1138 **quired]** target group Object.

1139 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1140 **[Optional]** that the PS provider send a notification if and when the membership of the targeted group
1141 changes.

1142 The schema declaration for the <AddToCollectionRequest> message is shown below.

```
1143
1144 <!-- Declaration of AddToCollectionRequest element -->
1145 <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>
1146 <!-- Definition of AddToCollectionRequestType -->
1147 <xs:complexType name="AddToCollectionRequestType">
1148   <xs:complexContent>
1149     <xs:extension base="RequestAbstractType">
1150       <xs:sequence>
1151         <xs:element ref="TargetObjectID"/>
1152         <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>
1153         <xs:element ref="Subscription" minOccurs="0"/>
1154       </xs:sequence>
1155     </xs:extension>
1156   </xs:complexContent>
1157 </xs:complexType>
1158
```

1159 The following is an example of an <AddToCollectionRequest> message used to add three Objects to the target
1160 group Object.

```
1161
1162 <AddToCollectionRequest>
1163   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
1164   <ObjectID>https://ps.com/qaf3eflo</ObjectID>
1165   <ObjectID>https://ps.com/bzpfnrns</ObjectID>
1166   <ObjectID>https://ps.com/nsdf1fss</ObjectID>
```

1167 </AddToCollectionRequest>
1168

1169 3.14.3. AddToCollectionResponse Message

1170 A PS provider responds to an <AddToCollectionRequest> message with an <AddToCollectionResponse>
1171 message. The PS provider makes the indicated modification to the specified target <Object> and responds with the
1172 status.

1173 The <AddToCollectionResponse> message has the type of **ResponseAbstractType**

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

```
1175  
1176 <!-- Declaration of AddToCollectionResponse element -->  
1177 <xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>  
1178
```

1179 The following is an example of an <AddToCollectionResponse> to the <AddToCollectionRequest> message
1180 above. The PS provider is responding that the request that the three Objects be added to the target Object was
1181 successful.

```
1182  
1183 <AddToCollectionResponse>  
1184   <Status code="OK"/>  
1185 </AddToCollectionResponse>  
1186
```

1187 3.14.4. Processing Rules

1188 The WSC:

1189 • MUST specify, as a value of the <TargetObjectID>, an ObjectID of the Object that has *urn:liberty:ps:*
1190 *collection* as a value of *NodeType* attribute.

1191 The PS provider:

1192 • MUST, if the Object specified by the value of the <TargetObjectID> element has *urn:liberty:ps:entity* as the
1193 value of its *NodeType* attribute, respond with *Failed* as the code attribute of the top level <lu:Status> **element**.
1194 **A second level <lu:Status> MAY be inserted. If and so, the code attribute of that second level <lu:Status>**
1195 **element MUST be set with the following status code:**

1196 • ObjectIsEntity

1197 • **MUST, if the Object specified by the value of an <ObjectID> element is already a member of the targeted**
1198 **collection, respond with Failed as the code attribute of the top level <lu:Status> element. A second level <lu:**
1199 **Status> MAY be inserted. If so, the code attribute of that second level <lu:Status> element MUST be set with**
1200 **the following status code:**

1201 • DuplicateObject

1202 • **MUST NOT allow the creation of circular collections. A circular collection is one which includes, at any layer of**
1203 **the structure below it, a reference to the same collection such that a dereference of the collection would result in**
1204 **an infinite loop. If the PS provider receives an <AddToCollectionRequest> message that would result in a**
1205 **circular collection, it MUST respond with Failed as the code attribute of the top level <lu:Status> element. A**
1206 **second level <lu:Status> MAY be inserted. If so, the code attribute of that second level <lu:Status> element**
1207 **MUST be set with the following status code:**

1208 • [CircularCollection](#)

1209 **3.15. Removing from a Collection**

1210 A WSC uses the <RemoveFromCollectionRequest> message to request the removal of a child Object element
1211 from a parent group Object. Both user and group Objects can be removed from a parent group Object with the
1212 <RemoveFromCollectionRequest> message.

1213 **3.15.1. wsa:Action Values**

1214 <RemoveFromCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1215 "urn:liberty:ps:2006-08:RemoveFromCollectionRequest." <RemoveFromCollectionResponse> messages
1216 MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:RemoveFromCollection-
1217 Response."

1218 **3.15.2. RemoveFromCollectionRequest Message**

1219 The target parent group Object from which a child Object is to be removed is indicated by the value of the
1220 <TargetObjectID> element within the <RemoveFromCollectionRequest> message. The child Object being
1221 removed are specified by the value(s) of the <ObjectID> elements within the
1222 <RemoveFromCollectionRequest> element.

1223 The <RemoveFromCollectionRequest> message has the complex type
1224 **RemoveFromCollectionRequestType**, which extends **RequestAbstractType** and adds the following ele-
1225 ments:

1226 <TargetObjectID> The <TargetObjectID> element is used to convey the ObjectID of the target group
1227 **[Required]** Object from which specified Objects are to be removed.

1228 <ObjectID> **[Re-** The <ObjectID> element is used to convey ObjectIDs of the Objects that would be re-
1229 **quired]** moved from the target group Object.

1230 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1231 **[Optional]** that the PS provider send a notification if and when the membership of the targeted group
1232 changes.

1233 The schema declaration for the <RemoveFromCollectionRequest> message is shown below.

```
1234
1235 <!-- Declaration of RemoveFromCollectionRequest element -->
1236 <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
1237 <!-- Definition of RemoveFromCollectionRequestType -->
1238 <xs:complexType name="RemoveFromCollectionRequestType">
1239   <xs:complexContent>
1240     <xs:extension base="RequestAbstractType">
1241       <xs:sequence>
1242         <xs:element ref="TargetObjectID"/>
1243         <xs:element ref="ObjectID" maxOccurs="unbounded"/>
1244         <xs:element ref="Subscription" minOccurs="0"/>
1245       </xs:sequence>
1246     </xs:extension>
1247   </xs:complexContent>
1248 </xs:complexType>
1249
```

1250 The following is an example of a <RemoveFromCollectionRequest> message used to remove three Objects from
1251 the target group Object.

```
1252
1253 <RemoveFromCollectionRequest>
```

```
1254     <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
1255     <ObjectID>https://ps.com/qaf3eflo</ObjectID>
1256     <ObjectID>https://ps.com/bzpfnrns</ObjectID>
1257     <ObjectID>https://ps.com/nsdfllfss</ObjectID>
1258 </RemoveFromCollectionRequest>
1259
```

1260 3.15.3. RemoveFromCollectionResponse Message

1261 A PS provider responds to a <RemoveFromCollectionRequest> message with a
1262 <RemoveFromCollectionResponse> message. The PS provider makes the indicated modification to the specified
1263 target Object and responds with the status.

1264 The <RemoveFromCollectionResponse> message has the type of **ResponseAbstractType**

1265 The schema declaration for the <RemoveFromCollectionResponse> message is shown below.

```
1266
1267 <!-- Declaration of RemoveFromCollectionResponse element -->
1268 <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>
1269
```

1270 The following is an example of a <RemoveFromCollectionResponse> to the
1271 <RemoveFromCollectionRequest> message above. The PS provider is responding that the request that the three
1272 objects be removed from the target Object was successful.

```
1273
1274 <RemoveFromCollectionResponse>
1275     <Status code="OK"/>
1276 </RemoveFromCollectionResponse>
1277
```

1278 3.15.4. Processing Rules

1279 The WSC:

- 1280 • MUST specify, as a value of the <TargetObjectID>, an ObjectID of the Object that has *urn:liberty:ps:*
1281 *collection* as a value of Node Type attribute.

1282 The PS provider:

- 1283 • MUST, if the Object specified by the value of the <TargetObjectID> element has *urn:liberty:ps:entity* as the
1284 value of its Node Type attribute, respond with *Failed* as the code attribute of the top level <lu:Status> **element**.
1285 **A second level <lu:Status> MAY be inserted. If so, and the code attribute of that second level <lu:Status>**
1286 **element MUST be set with the following status code:**

- 1287 • ObjectIsEntity

- 1288 • **MUST, if the Object specified by the value of an <ObjectID> element is not a member of the targeted collection,**
1289 **respond with Failed as the code attribute of the top level <lu:Status> element. A second level <lu:Status>**
1290 **MAY be inserted. If so, the code attribute of that second level <lu:Status> element MUST be set with the**
1291 **following status code:**

- 1292 • CannotFindObject

1293 3.16. Listing Members

1294 A WSC uses the <ListMembersRequest> message to navigate the Object structure of the users (*urn:liberty:ps:*
1295 *entity* Objects) and groups (*urn:liberty:ps:collection* Objects) that comprise the PS resource.

1296 A WSC can control how Objects are returned by specifying its preferences with the Structured attribute.

1297 If a WSC does not specify a <TargetObjectID> element in the <ListMembersRequest> message, this is equivalent
1298 to asking for all top-level Object element, i.e. the default targeted Object is the root node.

1299 3.16.1. wsa:Action Values

1300 <ListMembersRequest> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:
1301 ps:2006-08:ListMembersRequest." <ListMembersResponse> messages MUST include a <wsa:Action> SOAP
1302 header with the value of "urn:liberty:ps:2006-08:ListMembersResponse."

1303 3.16.2. ListMembersRequest Message

1304 The WSC indicates to the PS provider the Object of interest by specifying that <Object> element's ObjectID in
1305 the <TargetObjectID> element. If no <TargetObjectID> element is specified in the <ListMembersRequest>
1306 message, the PS provider MUST return all the top-level Objects (i.e., Objects that do not have any parent
1307 Object.)

1308 The <ListMembersRequest> message has the complex type **ListMembersRequestType**, which extends
1309 **RequestAbstractType** and adds the following attributes and elements:

1310 Structured [Op- The Structured allows a WSC to indicate what portion of the Object tree structure it
1311 tional] desires be returned. See [Section 3.16.2.1](#) for more detail.

1312 Count [Optional] The Count attribute specifies how many child <Object> elements should be returned in a
1313 <ListMembersResponse> message. See [Section 3.16.2.2](#) for more detail.

1314 Offset [Optional] The Offset attribute specifies from which element to continue, when requesting more data.
1315 See [Section 3.16.2.2](#) for more detail.

1316 <TargetObjectID> The <TargetObjectID> element is used to convey an ObjectID of the target group
1317 [Optional] Object whose child Objects are to be listed.

1318 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1319 [Optional] that the PS provider send a notification if and when the membership of the targeted group
1320 changes.

1321 Interpretation of the membership for which a change <Notify> message should be sent will
1322 depend on the value of the Structured attribute on the <ListMembersRequest> message.
1323 For instance, if the attribute has a value of "tree," the subscription corresponds to the full
1324 object tree and <Notify> messages MUST be sent accordingly.

1325 For a subscription within a <ListMembersRequest> message, the PS provider MUST assess
1326 "changes" against whatever was returned in the original <ListMembersResponse>. A
1327 <Notify> MUST be sent if, were the WSC to resend the same request, the results would be
1328 different than originally sent.

1329 The schema declaration for the <ListMembersRequest> message is shown below.

```
1330
1331 <!-- Declaration of ListMembersRequest element -->
1332 <xs:element name="ListMembersRequest" type="ListMembersRequestType" />
1333 <!-- Definition of ListMembersRequestType -->
1334 <xs:complexType name="ListMembersRequestType">
1335     <xs:complexContent>
1336         <xs:extension base="RequestAbstractType">
1337             <xs:sequence>
1338                 <xs:element ref="TargetObjectID" minOccurs="0" />
```

```

1339         <xs:element ref="Subscription" minOccurs="0"/>
1340     </xs:sequence>
1341     <xs:attribute name="Structured" type="xs:string" use="optional"/>
1342     <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
1343     <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
1344 </xs:extension>
1345 </xs:complexContent>
1346 </xs:complexType>
1347

```

1348 3.16.2.1. Structured Attribute

1349 WSCs may choose to navigate the hierarchal tree structure for a given Object incrementally (i.e., by successive
1350 requests to probe deeper) or to have the complete tree returned. The Structured attribute allows the WSC to indicate
1351 this preference to the PS provider. Additionally, the WSC uses the Structured attribute to indicate whether or not it
1352 desires to see collection Object elements returned or only entity Object elements.

1353 The Structured attribute takes three possible values

1354 *children* The WSC is indicating that it desires only the direct child collection and entity objects of the
1355 targeted object.objects.

1356 The default value for the Structured attribute is *children*.

1357 *tree* the WSC is indicating that it desires the full tree structure of the targetted object.

1358 *entities* the WSC is indicating that it desires a flat view of the full tree structure, e.g., one in which any
1359 collection hierarchy is hidden.

1360 3.16.2.2. Count and Offset Attributes

1361 When the specified Object has multiple child Objects, the WSC may desire to be sent the child <Object> elements
1362 in smaller sets (i.e., a smaller number of elements at a time). This is achieved by using the Count and Offset attributes
1363 of the <ListMembersRequest> element.

1364 The Count attribute defines how many child <Object> elements should be returned in a
1365 <ListMembersResponse> message. The Count attribute only pertains to the direct child <Object> elements of the
1366 Object specified in the <ListMembersRequest> message.

1367 The Offset attribute specifies from which element to continue, when requesting for more data. The default value is
1368 zero, which refers to the first child <Object> element.

1369 3.16.3. ListMembersResponse Message

1370 A PS provider responds to a <ListMembersRequest> message with a <ListMembersResponse> message con-
1371 taining the appropriate set of <Object> elements.

1372 The <ListMembersResponse> message has the complex type **ListMembersResponseType**, which extends
1373 **ResponseAbstractType** and adds the following element:

1374 <Object> [**Optional**] The <Object> element is used to convey data of zero or more Objects to be listed.

1375 The schema declaration for the <ListMembersResponse> message is shown below.

```

1376
1377 <!-- Declaration of ListMembersResponse element -->
1378 <xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
1379 <!-- Definition of ListMembersResponseType -->
1380 <xs:complexType name="ListMembersResponseType">

```

```

1381     <xs:complexContent>
1382       <xs:extension base="ResponseAbstractType">
1383         <xs:sequence>
1384           <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
1385         </xs:sequence>
1386       </xs:extension>
1387     </xs:complexContent>
1388 </xs:complexType>
1389

```

1390 3.16.4. Examples

1391 All the examples described in this subsection assume that a PS provider has the following virtual XML document for
 1392 some user's PS list.

```

1393
1394 <Object NodeType="urn:liberty:ps:entity">
1395   <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1396   <DisplayName>Mary</DisplayName>
1397 </Object>
1398 <Object NodeType="urn:liberty:ps:entity">
1399   <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1400   <DisplayName>Bob</DisplayName>
1401 </Object>
1402 <Object NodeType="urn:liberty:ps:entity">
1403   <ObjectID>https://ps.com/reremvls</ObjectID>
1404   <DisplayName>Nick</DisplayName>
1405 </Object>
1406 <Object NodeType="urn:liberty:ps:entity">
1407   <ObjectID>https://ps.com/soijfsfd</ObjectID>
1408   <DisplayName>JoJo</DisplayName>
1409 </Object>
1410 <Object NodeType="urn:liberty:ps:entity">
1411   <ObjectID>https://ps.com/sdosafms</ObjectID>
1412   <DisplayName>Taro</DisplayName>
1413 </Object>
1414 <Object NodeType="urn:liberty:ps:entity">
1415   <ObjectID>https://ps.com/lgsdfsfd</ObjectID>
1416   <DisplayName>Hanako</DisplayName>
1417 </Object>
1418 <Object NodeType="urn:liberty:ps:collection">
1419   <ObjectID>https://ps.com/eiruvoie</ObjectID>
1420   <DisplayName>Soccer Team</DisplayName>
1421   <Object NodeType="urn:liberty:ps:collection">
1422     <ObjectID>https://ps.com/nmerflas</ObjectID>
1423     <DisplayName>Starting Members</DisplayName>
1424     <ObjectRef Ref="https://ps.com/lsdjfojd"/>
1425     <ObjectRef Ref="https://ps.com/sijfsfsf"/>
1426   </Object>
1427     <ObjectRef Ref="https://ps.com/reremvls"/>
1428     <ObjectRef Ref="https://ps.com/soijfsfd"/>
1429   </Object>
1430 <Object NodeType="urn:liberty:ps:collection">
1431   <ObjectID>https://ps.com/zxlidfdf</ObjectID>
1432   <DisplayName>Family</DisplayName>
1433   <ObjectRef Ref="https://ps.com/sdosafms"/>
1434   <ObjectRef Ref="https://ps.com/lgsdfsfd"/>
1435 </Object>
1436

```

1437 The following is an example of a <ListMembersRequest> message by which a WSC requests the list of members
 1438 of the "Soccer Team" collection Object. As the WSC specifies Structured="entities", it is indicating that it
 1439 desires to have a "flat" view of that group's Object tree returned, e.g., one in which all collection hierarchy is removed. |

```

1440
1441 <ListMembersRequest Structured="entities">

```

```

1442     <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1443 </ListMembersRequest>
1444

```

1445 The following is an example `<ListMembersResponse>` message as might be returned to the
 1446 `<ListMembersRequest>` above.

```

1447
1448 <ListMembersResponse>
1449   <Status code="OK"/>
1450   <Object NodeType="urn:liberty:ps:entity">
1451     <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1452     <DisplayName>Mary</DisplayName>
1453   </Object>
1454   <Object NodeType="urn:liberty:ps:entity">
1455     <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1456     <DisplayName>Bob</DisplayName>
1457   </Object>
1458   <Object NodeType="urn:liberty:ps:entity">
1459     <ObjectID>https://ps.com/reremvls</ObjectID>
1460     <DisplayName>Nick</DisplayName>
1461   </Object>
1462   <Object NodeType="urn:liberty:ps:entity">
1463     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1464     <DisplayName>JoJo</DisplayName>
1465   </Object>
1466 </ListMembersResponse>
1467

```

1468 As the WSC indicated it desired a flat view, the PS expanded the group `Object` called "Starting Members" and returns
 1469 its two child entity `Object` elements (for Mary and Bob) instead of the collection `Object` itself.

1470 Alternatively, the following is a `<ListMembersResponse>` message as might be returned to a
 1471 `<ListMembersRequest>` message in which the WSC indicated it desired to see the full tree structure by specifying
 1472 `Structured="tree"`.

```

1473
1474 <ListMembersResponse>
1475   <Status code="OK"/>
1476   <Object NodeType="urn:liberty:ps:collection">
1477     <ObjectID>https://ps.com/nmerflas</ObjectID>
1478     <DisplayName>Starting Members</DisplayName>
1479     <Object NodeType="urn:liberty:ps:entity">
1480       <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1481       <DisplayName>Mary</DisplayName>
1482     </Object>
1483     <Object NodeType="urn:liberty:ps:entity">
1484       <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1485       <DisplayName>Bob</DisplayName>
1486     </Object>
1487   </Object>
1488   <Object NodeType="urn:liberty:ps:entity">
1489     <ObjectID>https://ps.com/reremvls</ObjectID>
1490     <DisplayName>Nick</DisplayName>
1491   </Object>
1492   <Object NodeType="urn:liberty:ps:entity">
1493     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1494     <DisplayName>JoJo</DisplayName>
1495   </Object>
1496 </ListMembersResponse>
1497

```

1498 The PS returns the full sub-tree for the specified target object including the hierarchy of the "Starting Members" group.

1499 Lastly, the following is an example <ListMembersResponse> message as might be returned to a
 1500 <ListMembersRequest> message in which the WSC indicated it desired to see only direct children by specifying
 1501 Structured="children".

```

1502
1503 <ListMembersResponse>
1504   <Status code="OK" />
1505   <Object NodeType="urn:liberty:ps:collection">
1506     <ObjectID>https://ps.com/nmerflas</ObjectID>
1507     <DisplayName>Starting Members</DisplayName>
1508   </Object>
1509   <Object NodeType="urn:liberty:ps:entity">
1510     <ObjectID>https://ps.com/reremvls</ObjectID>
1511     <DisplayName>Nick</DisplayName>
1512   </Object>
1513   <Object NodeType="urn:liberty:ps:entity">
1514     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1515     <DisplayName>JoJo</DisplayName>
1516   </Object>
1517 </ListMembersResponse>
1518
    
```

1519 3.16.5. Processing Rules

- 1520 • The WSC SHOULD NOT include a <Subscription> element in a <ListMembersRequest> message if the
 1521 Offset attribute has any value other than "0."

- 1522 • The PS provider MUST, if the <ListMembersRequest> contains a Subscription and the Offset attribute has any
 1523 value other than "0," and it is otherwise capable of returning results, return those results as indicated by the Count
 1524 and Offset attributes, but still reject the Subscription by responding with "OKButNoSubscription" as the code
 1525 attribute of the top level <lu:Status> element. A second level <lu:Status> MAY be inserted. If so, the code
 1526 attribute of that second level <lu:Status> element MUST be set with the following status code:
 - 1527 • NoSubscribeWithOffset

- 1528 • The PS provider SHOULD dereference all <ObjectRef> elements and replace them with the appropriate
 1529 <Object> elements before returning.

- 1530 • At any one level of the tree, the PS provider SHOULD remove all duplicate <Object> elements before returning.

- 1531 • If the Structured attribute is set as *tree*, the PS provider MUST return all the direct child and descendant
 1532 <Object> elements of the specified Object.

- 1533 • If the Structured attribute is not set in the request, the PS provider MUST process it as if it is set as *children*.

- 1534 • If the Structured attribute is set as *entities*, the PS provider MUST return all the direct child and descendant
 1535 entity <Object> elements of the specified Object (subject to the restriction defined by the Count attribute, if-
 1536 present). Any collection <Object> elements MUST be removed and only entity <Object> elements returned.

- 1537 • If the Structured attribute is set as *children*, the PS provider MUST return all the direct child collection and
 1538 entity <Object> elements of the specified Object.

- 1539 • If the Count attribute is included in a <ListMembersRequest> message, the PS provider SHOULD NOT respond
 1540 with more objects than specified. A PS provider MAY return a smaller number of objects than specified by the
 1541 Count attribute.

- 1542 • If the Offset attribute is included in a <ListMembersRequest> message, the PS provider SHOULD respond with
 1543 a list of <Object> elements, the first in the list being that <Object> element whose position in the complete list
 1544 is specified by the value of the Offset attribute.

- 1545 • If a PS provider receives a `<ListMembersRequest>` message on which the value of `<TargetObjectID>` matches
1546 that of an `<ObjectID>` element of a given Object, and if the value of the `NodeType` attribute of the matched
1547 Object is `urn:liberty:ps:entity`, then the PS provider MUST respond with *Failed* as the code attribute of the top
1548 level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, ~~and~~ the code attribute of that
1549 second level `<lu:Status>` element MUST be set with the following status code:
- 1550 • `ObjectIsEntity`
- 1551 • The PS Provider MUST, if unable to find the targeted Object, respond with *Failed* as the code attribute of the top
1552 level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, ~~and~~ the code attribute of that
1553 second level `<lu:Status>` element MUST be set with the following status code:
- 1554 • `CannotFindObject`
- 1555 • When the targeted Object has no child Objects, the PS provider MUST respond with *OK* as the code attribute of
1556 the top level `<lu:Status>` element with no `<Object>` element in the response.
- 1557 • The PS provider SHOULD, if the `<ListMembersRequest>` contains a Subscription and the Structured attribute
1558 has any value other than "children," and it is otherwise capable of returning results, return those results but still
1559 reject the Subscription by responding with "OKButNoSubscription" as the code attribute of the top level `<lu:
1560 Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level
1561 `<lu:Status>` element MUST be set with the following status code:
- 1562 • `SubscribeToChildrenOnly`
- 1563 An "OKButNoSubscription" Status code SHOULD NOT be interpreted as reflecting a PS provider's overall ability
1564 to support subscriptions, rather simply its unwillingness to accept the particular subscription requested.

1565 3.17. Retrieving Info

1566 A WSC uses the `<GetObjectInfoRequest>` message to retrieve information for a particular Object. An
1567 Object's information includes all the child elements and attributes of the `<Object>` element, except for either
1568 `<Object>` or `<ObjectRef>` elements.

1569 Note that if a WSC wants to get a child *members'* Objects of the particular Object, the WSC MUST use
1570 `<ListMembersRequest>` message (see Section 3.16).

1571 3.17.1. wsa:Action Values

1572 `<GetObjectInfoRequest>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liber-
1573 ty:ps:2006-08:GetObjectInfoRequest." `<GetObjectInfoResponse>` messages MUST include a `<wsa:Action>`
1574 SOAP header with the value of "urn:liberty:ps:2006-08:GetObjectInfoResponse."

1575 3.17.2. GetObjectInfoRequest Message

1576 The WSC indicates to the PS provider the Object of interest by specifying that `<Object>` element's ObjectID in
1577 the `<TargetObjectID>` element.

1578 The `<GetObjectInfoRequest>` message has the complex type `GetObjectInfoRequestType`, which extends
1579 `RequestAbstractType` and adds the following element:

1580 `<TargetObjectID>` The `<TargetObjectID>` element is used to convey an ObjectID of the target Object of
1581 [Required] which information is requested.

1582 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1583 **[Optional]** that the PS provider send a notification if and when the information (but not membership) of
1584 the targeted Object changes.

1585 The schema declaration for the <GetObjectInfoRequest> message is shown below.

```
1586
1587 <!-- Declaration of GetObjectInfoRequest element -->
1588 <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
1589 <!-- Definition of GetObjectInfoRequestType -->
1590 <xs:complexType name="GetObjectInfoRequestType">
1591   <xs:complexContent>
1592     <xs:extension base="RequestAbstractType">
1593       <xs:sequence>
1594         <xs:element ref="TargetObjectID"/>
1595         <xs:element ref="Subscription" minOccurs="0"/>
1596       </xs:sequence>
1597     </xs:extension>
1598   </xs:complexContent>
1599 </xs:complexType>
1600
```

1601 The following is an example of a <GetObjectInfoRequest> message.

```
1602
1603 <GetObjectInfoRequest>
1604   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1605 </GetObjectInfoRequest>
1606
```

1607 **3.17.3. GetObjectInfoResponse Message**

1608 A PS provider responds to a <GetObjectInfoRequest> message with a <GetObjectInfoResponse> message,
1609 in which the specified Object's information is conveyed.

1610 The <GetObjectInfoResponse> message has the complex type **GetObjectInfoResponseType**, which extends
1611 **ResponseAbstractType** and adds the following elements:

1612 <Object> **[Optional]** The <Object> element is used to convey the information of the requested Object.

1613 The schema declaration for the <GetObjectInfoResponse> message is shown below.

```
1614
1615 <!-- Declaration of GetObjectInfoResponse element -->
1616 <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
1617 <!-- Definition of GetObjectInfoResponseType -->
1618 <xs:complexType name="GetObjectInfoResponseType">
1619   <xs:complexContent>
1620     <xs:extension base="ResponseAbstractType">
1621       <xs:sequence>
1622         <xs:element ref="Object" minOccurs="0"/>
1623       </xs:sequence>
1624     </xs:extension>
1625   </xs:complexContent>
1626 </xs:complexType>
1627
```

1628 The following is an example of a <GetObjectInfoResponse> to the <GetObjectInfoRequest> message above.

```
1629
1630 <GetObjectInfoResponse>
1631   <Status code="OK"/>
1632   <Object NodeType="urn:liberty:ps:collection">
1633     <ObjectID>https://ps.com/eiruvoie</ObjectID>
```

```
1634         <DisplayName>Soccer Team</DisplayName>
1635     </Object>
1636 </GetObjectInfoResponse>
1637
```

1638 3.17.4. Processing Rules

1639 A PS provider:

1640 • MUST return the <Object> corresponding to the <TargetObjectID> element on the
1641 <GetObjectInfoRequest> message.

1642 • MUST, if it can not find the **targeted** Object, respond with *Failed* as the code attribute of the top level <lu:
1643 Status> **element**. A second level <lu: Status> **MAY** be inserted. If so, ~~and~~ the code attribute of **that** second level
1644 <lu: Status> element **MUST** be set with the following status code:

1645 • CannotFindObject

1646 • MUST NOT respond with any child <Object> and/or <ObjectRef> elements of the specified <Object>.

1647 3.18. Updating Info

1648 A WSC may wish to update or modify the information for an Object, e.g., to change a DisplayName etc.

1649 A WSC uses the <SetObjectInfoRequest> message to update the information for a particular Object. Updateable
1650 information does not include <Object> element, <ObjectRef> element, NodeType attribute, CreatedDateTime
1651 attribute, and ModifiedDateTime attribute.

1652 Note that if a WSC wants to insert a child Object to the particular Object, the WSC MUST use
1653 <AddToCollectionRequest> message (see [Section 3.14](#)). Also note that if a WSC wants to remove a child
1654 Object from the particular Object, the WSC MUST use <RemoveFromCollectionRequest> message (see [Sec-
1655 tion 3.15](#)).

1656 3.18.1. wsa:Action Values

1657 <SetObjectInfoRequest> messages MUST include a <wsa: Action> SOAP header with the value of "**urn: liber-
1658 ty: ps: 2006-08: SetObjectInfoRequest.**" <SetObjectInfoResponse> messages MUST include a <wsa: Action>
1659 SOAP header with the value of "**urn: liberty: ps: 2006-08: SetObjectInfoResponse.**"

1660 3.18.2. SetObjectInfoRequest Message

1661 The WSC specifies <Object> elements of the target Object for updating. These <Object> elements MUST NOT
1662 have child <Object> and/or <ObjectRef> elements. Also, these <Object> elements MUST NOT have
1663 CreatedDateTime and ModifiedDateTime attributes.

1664 The <SetObjectInfoRequest> message has the complex type **SetObjectInfoRequestType**, which extends
1665 **RequestAbstractType** and adds the following element:

1666 <Object> [**Re-** The <Object> element is used to convey the updated data of the Object to be updated. **The**
1667 **quired**] <Object> element **MUST** have an <ObjectID> element that identifies the relevant object.

1668 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1669 [**Optional**] that the PS provider send a notification if and when the information of the targeted Object
1670 changes.

1671 The schema declaration for the <SetObjectInfoRequest> message is shown below.

```

1672
1673 <!-- Declaration of SetObjectInfoRequest element -->
1674 <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
1675 <!-- Definition of SetObjectInfoRequestType -->
1676 <xs:complexType name="SetObjectInfoRequestType">
1677   <xs:complexContent>
1678     <xs:extension base="RequestAbstractType">
1679       <xs:sequence>
1680         <xs:element ref="Object" maxOccurs="unbounded"/>
1681         <xs:element ref="Subscription" minOccurs="0"/>
1682       </xs:sequence>
1683     </xs:extension>
1684   </xs:complexContent>
1685 </xs:complexType>
1686

```

1687 The following is an example of a <SetObjectInfoRequest> message in which the WSC is changing the value of
 1688 the <DisplayName> element for the specified <Object>. The previously existing value for this element would be
 1689 overwritten.

```

1690
1691 <SetObjectInfoRequest>
1692   <Object NodeType="urn:liberty:ps:collection">
1693     <ObjectID>https://ps.com/eiruvoie</ObjectID>
1694     <DisplayName>Baseball Team</DisplayName>
1695   </Object>
1696 </SetObjectInfoRequest>
1697

```

1698 3.18.3. SetObjectInfoResponse Message

1699 A PS provider responds to a <SetObjectInfoRequest> message with a <SetObjectInfoResponse> message.

1700 The <SetObjectInfoResponse> message has the type of **ResponseAbstractType**

1701 The schema declaration for the <SetObjectInfoResponse> message is shown below.

```

1702
1703 <!-- Declaration of SetObjectInfoResponse element -->
1704 <xs:element name="SetObjectInfoResponse" type="ResponseAbstractType"/>
1705

```

1706 The following is an example of a <SetObjectInfoResponse> to the <SetObjectInfoRequest> message above.

```

1707
1708 <SetObjectInfoResponse>
1709   <Status code="OK"/>
1710 </SetObjectInfoResponse>
1711

```

1712 3.18.4. Processing Rules

1713 A PS provider:

- 1714 • MUST, if it can not find the target Object specified with the ObjectID, respond with *Failed* as the code attribute
 1715 of the top level <lu:Status> element. A second level <lu:Status> MAY be inserted. If so, and the code attribute
 1716 of that second level <lu:Status> element MUST be set with the following status code:

- 1717 • CannotFindObject

- 1718 • MUST, if it finds that the value of the specified NodeType attribute is different from the value of the NodeType
 1719 attribute of the target Object specified with the ObjectID, respond with *Failed* as the code attribute of the top

1720 level <lu: Status> **element**. A second level <lu: Status> **MAY** be inserted. If so, ~~and~~ the code attribute of that
1721 second level <lu: Status> element **MUST** be set with the following status code:

1722 • InvalidNodeType

1723 • **MUST**, if it finds that a WSC specifies a **child** ~~the~~ <Object> element, <ObjectRef> **elements**,
1724 CreatedDateTime attribute, or ModifiedDateTime, **attribute**, ignore these elements and/or attributes.

1725 3.19. Querying Objects

1726 A WSC may wish to have returned to it a list of Objects that meet some criteria. The <QueryObjectsRequest>
1727 message allows the WSC to indicate this of the PS provider. The criteria to be met are specified in the <Filter>
1728 element in the <QueryObjectsRequest> element.

1729 Note: The <ListMembersRequest> message can be considered a specialization of the
1730 <QueryObjectsRequest> message, in which the criteria to be met is that the returned Objects are members of the
1731 specified group Object. The <QueryObjectsRequest> message allows a WSC to pose more generalized queries,
1732 e.g., to which groups does a specific user belong?

1733 3.19.1. wsa:Action Values

1734 <QueryObjectsRequest> messages **MUST** include a <wsa: Action> SOAP header with the value of "**urn: liber-**
1735 **ty: ps: 2006-08: QueryObjectsRequest.**" <QueryObjectsResponse> messages **MUST** include a <wsa: Action>
1736 SOAP header with the value of "**urn: liberty: ps: 2006-08: QueryObjectsResponse.**"

1737 3.19.2. QueryObjectsRequest Message

1738 The WSC specifies criteria of its interest in the <Filter> element.

1739 The <QueryObjectsRequest> message has the complex type **QueryObjectsRequestType**, which extends
1740 **RequestAbstractType** and adds the following elements:

1741 <Filter> [**Re-** The <Filter> element is used to convey criteria for matching Object elements that a WSC
1742 **quired**] is interested in.

1743 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1744 [**Optional**] that the PS provider send a notification if and when the set of objects that satisfy the specified
1745 filter changes.

1746 Count [**Optional**] The Count attribute specifies how many <Object> elements should be returned in a
1747 <QueryObjectsResponse> message. See [Section 3.19.2.2](#) for more detail.

1748 Offset [**Optional**] The Offset attribute specifies from which element to continue, when requesting for more
1749 data. See [Section 3.19.2.2](#) for more detail.

1750 The schema declaration for the <QueryObjectsRequest> message is shown below.

```
1751
1752 <!-- Declaration of QueryObjectsRequest element -->
1753 <xs:element name="QueryObjectsRequest" type="QueryObjectsRequestType"/>
1754 <!-- Definition of QueryObjectsRequestType -->
1755 <xs:complexType name="QueryObjectsRequestType">
1756   <xs:complexContent>
1757     <xs:extension base="RequestAbstractType">
1758       <xs:sequence>
1759         <xs:element name="Filter" type="xs:string"/>
1760         <xs:element ref="Subscription" minOccurs="0"/>
1761       </xs:sequence>

```

```

1762         <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
1763         <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
1764     </xs:extension>
1765 </xs:complexContent>
1766 </xs:complexType>
1767

```

1768 3.19.2.1. Filter Element

1769 The value of the <Filter> element SHOULD be specified with an XPath expression.

1770 Any XPath expression MUST be evaluated against the set of Objects that would be returned to a hypothetical
 1771 <ListMembersRequest> that had targeted the root node and in which the Structured attribute was set to "tree."

1772 For instance, if the WSC wants to get all the <Object> elements with a NodeType attribute with a value of *urn:*
 1773 *liberty:ps:collection* (i.e., the WSC wants to get all the group Objects), the WSC can specify the value of the
 1774 <Filter> element as `"//ps:Object[@NodeType='urn:liberty:ps:collection']"`

1775 The following is an example of a <QueryObjectsRequest> message with which the WSC requests to get all the
 1776 <Object> elements that have *urn:liberty:ps:entity* as the value of NodeType element.

```

1777 <QueryObjectsRequest>
1778     <Filter>//ps:Object[@NodeType='urn:liberty:ps:entity']</Filter>
1779 </QueryObjectsRequest>
1780
1781

```

1782 3.19.2.2. Count and Offset Attributes

1783 When the result of specified criteria has multiple Objects, the WSC may desire to be sent their <Object> elements
 1784 in smaller sets (i.e., a smaller number of elements at a time). This is achieved by using the Count and Offset attributes
 1785 of the <QueryObjectsRequest> element.

1786 The Count attribute defines how many <Object> elements should be returned in a <QueryObjectsResponse>
 1787 message.

1788 The Offset attribute specifies from which element to continue, when requesting for more data. The default value is
 1789 zero, which refers to the first <Object> element.

1790 3.19.3. QueryObjectsResponse Message

1791 A PS provider responds to a <QueryObjectsRequest> message with a <QueryObjectsResponse> message. The
 1792 <QueryObjectsResponse> contains the <Object> elements that meet the criteria specified in the <Filter> ele-
 1793 ment of the <QueryObjectsRequest> message.

1794 The <QueryObjectsResponse> message has the complex type **QueryObjectsResponseType**, which extends
 1795 **ResponseAbstractType** and adds the following element:

1796 <Object> [**Optional**] The <Object> element is used to convey data of zero or more Objects that the requested
 1797 criteria are met to.

1798 The schema declaration for the <QueryObjectsResponse> message is shown below.

```

1799 <!-- Declaration of QueryObjectsResponse element -->
1800 <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
1801 <!-- Definition of QueryObjectsResponseType -->
1802 <xs:complexType name="QueryObjectsResponseType">
1803     <xs:complexContent>
1804         <xs:extension base="ResponseAbstractType">

```

```

1806         <xs:sequence>
1807             <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
1808         </xs:sequence>
1809     </xs:extension>
1810 </xs:complexContent>
1811 </xs:complexType>
1812
1813

```

1814 The following is an example of a <QueryObjectsResponse> to the <QueryObjectsRequest> message above.

```

1815 <QueryObjectsResponse>
1816   <Status code="OK"/>
1817   <Object NodeType="urn:liberty:ps:entity">
1818     <ObjectID>https://ps.com/lstdjfojd</ObjectID>
1819     <DisplayName>Mary</DisplayName>
1820   </Object>
1821   <Object NodeType="urn:liberty:ps:entity">
1822     <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1823     <DisplayName>Bob</DisplayName>
1824   </Object>
1825   <Object NodeType="urn:liberty:ps:entity">
1826     <ObjectID>https://ps.com/reremvls</ObjectID>
1827     <DisplayName>Nick</DisplayName>
1828   </Object>
1829   <Object NodeType="urn:liberty:ps:entity">
1830     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1831     <DisplayName>JoJo</DisplayName>
1832   </Object>
1833   <Object NodeType="urn:liberty:ps:entity">
1834     <ObjectID>https://ps.com/sdosafms</ObjectID>
1835     <DisplayName>Taro</DisplayName>
1836   </Object>
1837   <Object NodeType="urn:liberty:ps:entity">
1838     <ObjectID>https://ps.com/lgsdfsfd</ObjectID>
1839     <DisplayName>Hanako</DisplayName>
1840   </Object>
1841 </QueryObjectsResponse>
1842
1843

```

1844 3.19.4. Processing Rules

- 1845 • If a WSC specifies the value of the <Filter> element through an XPath expression, the value SHOULD begin
1846 with the expression **"//ps:Object."**

- 1847 • All the <Object> elements that are responded in the <QueryObjectsResponse> message from a PS provider
1848 MUST NOT contain any child <Object> and/or <ObjectRef> elements.

- 1849 • If a PS provider cannot find any Objects that meets the criteria that a WSC specifies in the request, the PS provider
1850 MUST respond *OK* as the code attribute of the top level <lu:Status> **element**. A second level <lu:Status>
1851 **MAY** and be inserted. If so, the code attribute of **that** second level <lu:Status> element MUST be set with the
1852 following status code:
 - 1853 • NoResults

- 1854 • A PS provider **MAY** choose to reject Subscriptions within <QueryObjectsRequest> messages (as might be
1855 desirable should the computational expense of determining when to send Notifications be prohibitive).

- 1856 The PS provider **MAY**, if the <QueryObjectsRequest> contained a Subscription it does not wish to accept and
1857 it is otherwise capable of returning results, return those results but still reject the Subscription by responding with
1858 "OKButNoSubscription" as the code attribute of the top level <lu:Status> element. A second level <lu:

1859 <Status> MAY be inserted. If so, the code attribute of that second level <lu: Status> element MUST be set with
1860 the following status code:

- 1861 • PolicyDoesNotAllow

1862 An "OKButNoSubscription" Status code SHOULD NOT be interpreted as reflecting a PS provider's overall ability
1863 to support subscriptions.

1864 3.20. Testing Membership

1865 A WSC may wish to pose a question of the Object tree structure and have the results of that question returned rather
1866 than the Object tree itself (as the <ListMembersRequest> or <QueryObjectsRequest> messages support). For
1867 instance, the WSC might wish to ask whether a specified individual (or more generally any Object) is a member of
1868 a particular specified group Object. This scenario will be common when the owning user has defined some access
1869 control policy in terms of membership in some group (e.g., "allow members of my soccer team to view these photos").
1870 If and when some other user tries to access the resource in question, the WSC will need to determine if they are entitled
1871 (e.g., whether or not they are on the soccer team).

1872 The <TestMembershipRequest> allows a WSC to pose the question "Is user X a member of group Y?"

1873 3.20.1. wsa:Action Values

1874 <TestMembershipRequest> messages MUST include a <wsa: Action> SOAP header with the value of "urn: lib-
1875 erty: ps: 2006-08: TestMembershipRequest." <TestMembershipResponse> messages MUST include a <wsa: Action>
1876 SOAP header with the value of "urn: liberty: ps: 2006-08: TestMembershipResponse."

1877 3.20.2. TestMembershipRequest Message

1878 The target parent group Object for which the membership of another Object is being tested is indicated by the value
1879 of the <TargetObjectID> element within the <TestMembershipRequest> message. The Object for which mem-
1880 bership is being tested is specified by the <Token> element within the <TestMembershipRequest> message.

1881 The <TestMembershipRequest> message has the complex type **TestMembershipRequestType**, which extends
1882 **RequestAbstractType** and adds the following elements:

1883 <TargetObjectID> The <TargetObjectID> element is used to convey the ObjectID of the target group
1884 **[Optional]** Object for which the membership of a user is being tested.

1885 Absence of the <TargetObjectID> indicates a request to test if the identity associated with
1886 the submitted token is associated to an Object (of type entity) in the PS.

1887 <Token> **[Required]** The <Token> element is used to convey an identity token of a user for which membership is
1888 being tested.

1889 <Subscription> The <Subscription> element is used to indicate to the PS provider that the WSC desires
1890 **[Optional]** that the PS provider send a notification if and when results of the test changes.

1891 The schema declaration for the <TestMembershipRequest> message is shown below.

```
1892
1893 <!-- Declaration of TestMembershipRequest element -->
1894 <xs:element name="TestMembershipRequest" type="TestMembershipRequestType" />
1895 <!-- Definition of TestMembershipRequestType -->
1896 <xs:complexType name="TestMembershipRequestType">
1897   <xs:complexContent>
1898     <xs:extension base="RequestAbstractType">
1899       <xs:sequence>
```

```

1900         <xs:element ref="TargetObjectID" minOccurs="0"/>
1901         <xs:element ref="Token"/>
1902         <xs:element ref="Subscription" minOccurs="0"/>
1903     </xs:sequence>
1904 </xs:extension>
1905 </xs:complexContent>
1906 </xs:complexType>

```

1907 The following is an example of a <TestMembershipRequest> message.

```

1908 <TestMembershipRequest>
1909   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1910   <Token>
1911
1912   </Token>
1913 </TestMembershipRequest>

```

1915 3.20.3. TestMembershipResponse Message

1916 The PS returns the result of the specified membership test in a <Result> element within a
1917 <TestMembershipResponse> message.

1918 The <TestMembershipResponse> message has the complex type **TestMembershipResponseType**, which extends
1919 **ResponseAbstractType** and adds the following elements:

1920 <Result> [**Re-** The <Result> element is used to convey the result of the specified membership test. This
1921 **quired**] element has a type of **ResultType**, which is derived from **xs:boolean**.

1922 The schema declarations for the <TestMembershipResponse> message and the <Result> element are shown below.

```

1923
1924 <!-- Definition of ResultType -->
1925 <xs:complexType name="ResultType">
1926   <xs:complexContent>
1927     <xs:extension base="xs:boolean"/>
1928   </xs:complexContent>
1929 </xs:complexType>
1930
1931 <!-- Declaration of TestMembershipResponse element -->
1932 <xs:element name="TestMembershipResponse" type="TestResponseType"/>
1933 <!-- Definition of TestMembershipResponseType -->
1934 <xs:complexType name="TestMembershipResponseType">
1935   <xs:complexContent>
1936     <xs:extension base="ResponseAbstractType">
1937       <xs:sequence>
1938         <xs:element name="Result" type="ResultType" minOccurs="0"/>
1939       </xs:sequence>
1940     </xs:extension>
1941   </xs:complexContent>
1942 </xs:complexType>

```

1943 The following is an example of a <TestMembershipResponse> message.

```

1944 <TestMembershipResponse>
1945   <Status code="OK"/>
1946   <Result>true</Result>
1947 </TestMembershipResponse>

```

1949 3.20.4. Processing Rules

1950 If the <TargetObjectID> element specifies the ObjectID of an Object with a NodeType attribute of "**urn:liber-**
1951 **ty:ps:entity**," the PS provider **MUST** respond with *Failed* as the code attribute of the top level <lu: Status> **element**.

1952 A second level `<lu: Status>` MAY be inserted. If so, and the code attribute of that second level `<lu: Status>` element
1953 MUST be set with the following status code:

- 1954 • ObjectIsEntity

1955 3.21. Resolving Objects

1956 Once a WSC has an `ObjectID` for an entity, it will often desire to communicate with other providers about that user.
1957 To do so, the WSC will first need an identity token for that user. The process of converting an object identifier into an
1958 identity token is referred to as *resolving* the identity token.

1959 3.21.1. wsa:Action Values

1960 `<ResolveIdentifierRequest>` messages MUST include a `<wsa: Action>` SOAP header with the value of "urn:
1961 liberty:ps:2006-08:ResolveIdentifierRequest." `<ResolveIdentifierResponse>` messages MUST include a
1962 `<wsa: Action>` SOAP header with the value of "urn:liberty:ps:2006-08:ResolveIdentifierResponse."

1963 3.21.2. ResolveIdentifierRequest Message

1964 The WSC can use the `<ResolveIdentifierRequest>` message to ask the PS provider to resolve the specified
1965 `ObjectID` in the `<TargetObjectID>` element, into an appropriate identity token.

1966 An `<ResolveIdentifierRequest>` message consists of one or more `<ResolveInput>` elements. If multiple
1967 `<ResolveInput>` elements are included in a request, then each MUST contain a `reqID` attribute so that the response
1968 contents can be correlated to them.

1969 The WSC MAY specify its requirements of the identity token through the `<sec: TokenPolicy>`

1970 The `<ResolveIdentifierRequest>` message has the complex type `ResolveIdentifierRequestType`, which
1971 extends `RequestAbstractType` and adds the following elements:

1972 `<ResolveInput>` The object for which the WSC desires an identity token be resolved.
[One or more]

1973 The schema declaration for the `<ResolveIdentifierRequest>` message is shown below.

```
1974
1975 <!-- Declaration of ResolveIdentifierRequest element -->
1976 <xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
1977 <!-- Definition of ResolveIdentifierRequestType -->
1978 <xs:complexType name="ResolveIdentifierRequestType">
1979   <xs:complexContent>
1980     <xs:extension base="RequestAbstractType">
1981       <xs:sequence>
1982         <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
1983       </xs:sequence>
1984     </xs:extension>
1985   </xs:complexContent>
1986 </xs:complexType>
1987
```

1988 3.21.2.1. ResolveInput element

1989 The `<ResolveInput>` element is of complex type `<ResolveInputType>`, which extends the complex type
1990 `<MappingInputType>` defined in [LibertyAuthn] to add the following element:

1991 `<TargetObjectID>` The `<TargetObjectID>` conveys the `ObjectID` of the target entity `Object` for which the
1992 [Required] WSC is requesting an identity token be resolved.

1993 The schema declaration for the <ResolveInput> element is shown below.

```

1994
1995 <!-- Declaration of ResolveInput element -->
1996 <xs:element name="ResolveInput" type="ResolveInputType"/>
1997 <!-- Definition of ResolveInputType -->
1998 <xs:complexType name="ResolveInputType">
1999     <xs:complexContent>
2000         <xs:extension base="ims:MappingInputType">
2001             <xs:sequence>
2002                 <xs:element ref="TargetObjectID"/>
2003             </xs:sequence>
2004         </xs:extension>
2005     </xs:complexContent>
2006 </xs:complexType>
2007

```

2008 The semantics and processing rules of the elements and attributes are as defined in [LibertyAuthn].

2009 The following is an example of a <ResolveIdentifierRequest> message in which the WSC is requesting that an
2010 identity token for the Object identified by the specified ObjectID be returned. The WSC is indicating that it desires
2011 a transient identifier for the identity token.

```

2012
2013 <ResolveIdentifierRequest>
2014     <ResolveInput>
2015         <TargetObjectID>https://ps.com/lgsdfsf</TargetObjectID>
2016         <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion"><sec:TokenPolicy>
2017             <samlp2:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
2018                 SPNameQualifier="https://psa.com"/></samlp2:NameIDPolicy>
2019         </sec:TokenPolicy>
2020     </ResolveInput>
2021 </ResolveIdentifierRequest>
2022

```

2023 3.21.3. ResolveIdentifierResponse Message

2024 A PS provider responds to a <ResolveIdentifierRequest> message with a <ResolveIdentifierResponse>
2025 message. The PS provider returns the identity tokens corresponding to the Objects specified by the ObjectID in the
2026 TargetObjectID element on the <ResolveIdentifierRequest> message.

2027 The schema declaration for the <ResolveIdentifierResponse> message is shown below.

```

2028
2029 <!-- Declaration of ResolveIdentifierResponse element -->
2030 <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType"/>
2031 <!-- Definition of ResolveIdentifierResponseType -->
2032 <xs:complexType name="ResolveIdentifierResponseType">
2033     <xs:complexContent>
2034         <xs:extension base="ResponseAbstractType">
2035             <xs:sequence>
2036                 <xs:element ref="ResolveOutput" maxOccurs="unbounded"/>
2037             </xs:sequence>
2038         </xs:extension>
2039     </xs:complexContent>
2040 </xs:complexType>
2041

```

2042 3.21.3.1. ResolveOutput element

2043 Each <ResolveOutput> element consists of an identity token (in the form of a <sec:Token>).

2044 The element is of complex type MappingOutputType defined in [LibertyAuthn]. Returned tokens are correlated with
2045 input object identifiers through the reference mechanism defined in [LibertyAuthn]

2046 The declaration for the <ResolveOutput> element is shown below.

```
2047
2048     <!-- Declaration of ResolveOutput element -->
2049     <xs:element name="ResolveOutput" type="ims:MappingOutputType" />
2050
```

2051 The following is an example of a <ResolveIdentifierResponse> to the <ResolveIdentifierRequest> mes-
2052 sage above.

```
2053
2054 <ResolveIdentifierResponse>
2055     <Status code="OK" />
2056     <ResolveOutput>
2057         <Token>
2058
2059         </Token>
2060     </ResolveOutput>
2061 </ResolveIdentifierResponse>
```

2062 3.21.4. Processing Rules

2063 • The PS provider **MUST**, if unable to find one or more (but not all) specified input objects, respond "PartialSuccess"
2064 as the code attribute of the top level <lu:Status> element, and **MAY** use second level <lu:Status> elements
2065 that containing a ref attribute equal to the associated <ResolveInput>'s reqID attribute. If unable to find any input
2066 objects, the PS provider **MUST** respond "Failed" as the code attribute of the top level <lu:Status> element. A
2067 second level <lu:Status> **MAY** be inserted. If so, the code attribute of that second level <lu:Status> element
2068 **MUST** be set with the following status code:

2069 • CannotFindObject

2070 • Upon receiving a <ResolveIdentifierRequest> from **an** SP carrying <TargetObjectID> elements that cor-
2071 respond to **an** existing Object, the PS provider **MUST** endeavor to return to the SP an appropriate identity token
2072 corresponding to that Object.

2073 To do so, the PS provider **MAY, itself**, send **an** <ims:IdentityMappingRequest> message to the relevant
2074 identity provider for the Object in question, specifying the requesting WSC as the target namespace.

2075 • If a PS provider ~~cannot~~**can not** resolve any of **the** input objects into identity tokens, the PS provider **MUST** respond
2076 **"Failed"** as the code attribute of the top level <lu:Status> element, and **SHOULD** use second level <lu:
2077 Status> elements that **MUST** contain a ref attribute equal to the associated <ResolveInput>'s reqID attribute.
2078 ~~The code attribute of any second level <lu:Status> element~~**elements corresponding to failed inputs** **MUST** be
2079 set with the following status code:

2080 • CannotResolveToken

2081 • If a PS provider is unable to resolve all input objects into identity tokens, **but is able to resolve some**, the PS provider
2082 **MUST** respond **"PartialSuccess"** as the code attribute of the top level <lu:Status> element, and **SHOULD** use
2083 second level <lu:Status> elements that **MUST** contain a ref attribute equal to the associated
2084 <ResolveInput>'s reqID attribute. **The code attribute of any second level <lu:Status> element** **MUST** be set
2085 **with the following status code:**

2086 • CannotResolveToken

2087 • The PS provider **MUST**, if one or more (but not all) specified input objects is of type "collection," respond "Par-
2088 tialSuccess" as the code attribute of the top level <lu:Status> element, and **SHOULD** use second level <lu:
2089 Status> elements that contain a ref attribute equal to the associated <ResolveInput>'s reqID attribute. If all
2090 input objects are of type "collection," the PS provider **MUST** respond "Failed" as the code attribute of the top level

- 2091 <lu:Status> element. The second level <lu:Status> elements corresponding to such failed inputs MUST be
2092 set with the following status code:
- 2093 • **ObjectIsCollection**

2094 4. Interaction with Users

2095 Before a user can be added to another user's PS resource, appropriate federations may need to be established between
2096 various providers. Before this can happen, it will typically be necessary for the user to *visit* these providers in order to
2097 kick-off the process. The mechanism by which this prompt occurs is referred to here as an *invitation*.

2098 Except for special cases where a user can be added to a user's PS resource without the participation of that user (as
2099 would be possible if the ~~PS-resource-owning~~PS-resource-owning user happened to know an identifier for the other user
2100 at some IDP), such an invitation is necessary. Supporting such user interactions is a key (but not mandatory) aspect of
2101 the PS role.

2102 Notwithstanding this, supporting the invitation model is optional from a conformance point of view (i.e., as defined
2103 by the [LibertyIDWSF20SCR]) and so any normative requirements expressed within this specification should be un-
2104 derstood in this context.

2105 4.1. Model (Informative)

2106 The invitation model is as follows:

- 2107 1. A user (visiting one of their SPs or their PS provider), decides that they wish to add some friend/contact/family
2108 member to their PS resource (likely in the context of enabling some specific interaction with that friend)
- 2109 2. An invitation (consisting of some human readable descriptive text explaining the context as well as some mech-
2110 anism by which interaction can be kicked off) is created by the provider on behalf of the PS resource owning user
2111 (the *inviting* user).
- 2112 3. The invitation is delivered to the relevant user (the *invited* user).
- 2113 4. The invited user examines the invitation and decides whether or not they wish to accept. If no, they take no further
2114 steps. If so, they proceed with the indicated mechanism to interact with the relevant providers (being given ap-
2115 propriate information and consent mechanisms at each step).
- 2116 5. The invited user is added to the inviting user's PS resource.
- 2117 6. The invited user can be directed to the SP to access the resource in question.

2118 It is also possible for a user to add their friends to their PS list through whatever browser-interface that PS provider
2119 makes available. In essence, these friends would be added independent of any particular interaction context, but avail-
2120 able for future interactions once added. In such a case there will still need to be an invitation sent to the friend being
2121 added (with the same proviso for a known identifier) in order to facilitate the establishment of a federation between
2122 the PS and the invited user's IDP, but the process can be simpler because there is no initiating SP.

2123 4.2. Additional Federations for Sharing of Identity Services

2124 When the invitation process is initiated from an SP and the resource being shared is browser-based, the normal result
2125 of successful interaction is that federated identifiers for the invited user are established between their IDP and both the
2126 PS provider and the initiating SP.

2127 When, rather than browser-based, the resource being shared is an identity service (e.g., a user's online presence) it may
2128 be necessary for an additional federation for the invited user be established between their IDP and the Discovery Service
2129 (DS) (see [LibertyDisco]) of the inviting user.

2130 Ultimately, when the shared resource is some identity service hosted by a WSP on behalf of one user (who has decided
2131 to make access to it available), it will be accessed by some WSC on behalf of the other user (that to which access
2132 privileges have been granted).

2133 To facilitate this operation, the DS of the inviting user will likely need to provide an appropriate WSP
2134 <EndpointReference> to the WSC upon that WSC querying for relevant (associated with the inviting user and of
2135 a particular service type) WSPs. If the inviting user's policy is such that merely the fact of existence or location of their
2136 identity data (beyond its actual value) warrants protection, then the DS should apply access control mechanisms to
2137 such WSC queries. To do so, it may need to have an identifier for the requesting user so that an appropriate access
2138 control decision can be made.

2139 In this sense, the existence and location of a user's identity services (as exemplified in the endpoint references pointing
2140 there), are identity resources like any other identity service and may require appropriate federations be established to
2141 enable access control.

2142 If policy is such that no such fine-grained access control for <EndpointReference>s is deemed necessary, then
2143 establishing a federation between the DS of the inviting user and the IDP of the invited user will not be necessary.

2144 **4.3. Consent Model**

2145 A number of consent models governing the various federations established through the invitation process are possible.
2146 A few examples, amongst others, are listed below:

2147 One possibility would be for an invited user, when federating their IDP to the PS provider of a friend upon an invitation
2148 from that friend, to specify that the establishment of subsequent federations between service providers and their IDP
2149 need not require additional consent. Such blanket consent, while optimizing usability, could present unforeseen risks,
2150 e.g., giving a user the permission, even if never taken advantage of, to view offensive online material.

2151 At the other extreme, a user could request that they be asked for specific consent for each and every such federation.

2152 Additionally, a user could specify that their IDP be allowed to establish "provisional" federated identifiers with other
2153 SPs, but that they expect to be given the opportunity to give explicit consent to these federations if and when they
2154 attempt to use them. The identifiers are provisional in the sense that the IDP will not actually agree to use them with
2155 the other provider until such consent is obtained, at which point they would become "real." This model would ensure
2156 that the user would not be presented with numerous requests for consent - consent would be obtained only when
2157 necessary.

2158 **4.4. Elements Supporting Invitation**

2159 The following elements support the invitation model. These elements are optional within the appropriate protocol
2160 messages.

2161 **4.4.1. PStoSPRedirectURL element**

2162 The SP MAY use the <PStoSPRedirectURL> element on <AddEntityRequest> and
2163 <AddKnownEntityRequest> messages to specify to the PS provider the URL (at the SP) to which that SP desires
2164 the invited user's user agent be directed after successful interaction and IDP federation has occurred. If and when the
2165 invited user's user agent has been sent to this URL, the SP MAY provide the invited user the designated access to the
2166 resource in question.

2167 The value of the <PStoSPRedirectURL> element MUST be such that, if and when a user agent is sent to this address
2168 from the PS provider, the SP can unambiguously determine the invitation to which the URL corresponds. The URL
2169 MUST be unique for each combination of the inviting user, the PS, and the invited user.

2170 **4.4.2. <SptoPSRedirectURL> element**

2171 The PS provider MAY use the <SptoPSRedirectURL> element on <AddEntityResponse> and
2172 <AddKnownEntityResponse> messages to specify to the SP the URL (at the PS provider) to which that PS desires
2173 the invited user's user agent be directed after successful initial interaction at the SP has occurred. If and when the invited

2174 user's user agent is sent to this URL, the PS provider will endeavour to establish a federation for that principal with the
2175 appropriate IDP.

2176 The value of the <SPtoPSredirectURL> element MUST be such that, if and when a user agent is sent to this address
2177 from the SP, the PS can unambiguously determine the invitation to which the URL corresponds. The URL MUST be
2178 unique for each combination of the inviting user (that owns the PS resource), the requesting SP, and the invited user.

2179 The PS provider MUST be prepared for the invited user to, at some point in the future, visit the URL provided in any
2180 specified <SPtoPSredirectURL> element. As it may be some time before the invited user does respond, the PS
2181 provider SHOULD store this url for a reasonable length of time.

2182 4.4.2.1. Processing Rules

2183 If and when the invited user responds to the invitation, the SP:

2184 • MUST, after appropriately informing and "consenting" the invited user, direct the user agent to the address previ-
2185 ously specified within the <SPtoPSredirectURL> element.

2186 Once the invited user has been redirected to the PS provider, the PS provider:

2187 • MUST determine the invited user's IDP (or preferred IDP if the invited user has multiple).

2188 MUST endeavor to establish a federated identity for that user with that IDP.

2189 • MAY obtain an identity token from the IDP for the invited user targeted for itself.

2190 • MUST redirect the invited user's agent to the address previously specified by any <PStoSPRedirectURL> element
2191 in the original <AddEntityRequest> or <AddKnownEntityRequest> message.

2192 Once the PS has redirected the invited user to the <PStoSPRedirectURL> address, the SP MAY choose to send a
2193 <samlp:AuthnRequest> message to the IDP asking for a <saml:AuthnStatement> attesting to that user's au-
2194 thentication status there.

2195 In its corresponding <samlp:Response>, the IDP SHOULD use the same subject identifier for this <saml:
2196 Assertion> as previously delivered to the SP within the identity token through the PS provider and the <Notify>
2197 message (unless any SP policy on the <samlp:AuthnRequest> message precludes this).

2198 4.4.3. <QueryString> element

2199 The <QueryString> element enables an alternative model for invited user interaction which is expected to better
2200 defend against identity theft attacks in which a valid email is spoofed to fool users into clicking an embedded URL.
2201 The invitation received by the invited user will contain a string carrying a SAML artifact (and potentially relay state
2202 info) representing a SAML <samlp:AuthnRequest> message created by the PS provider. The invited user can, if
2203 they choose, present this artifact string to their identity provider - which can then use the SAML <samlp:
2204 ArtifactResolve> message to retrieve the original <samlp:AuthnRequest> message from the PS provider.

2205 As the invited user visits their IDP by explicitly providing the address or using an existing bookmark, they can be more
2206 confident that the site is not spoofed. Once they are at their IDP and after presenting the SAML artifact, appropriate
2207 federations can be established for the invited user with the originating PS provider and SP.

2208 4.4.3.1. Schema

```
2209 <xs:element name="QueryString" type="QueryStringType"/>
2210   <xs:complexType name="QueryStringType">
2211     <xs:simpleContent>
2212       <xs:extension base="xs:string"/>
2213     </xs:simpleContent>
2214   </xs:complexType>
```

2215 4.4.3.2. Formatting Rules

2216 The contents of the <QueryString> element MUST satisfy the formatting requirements of the URL encoding of the
2217 SAML Artifact Binding (see [SAMLBind2]) as identified by the URI:

2218 *urn:oasis:names:tc:SAML:2.0:artifact-04.*

2219 The following is an example of an <QueryString> element in which the PS Provider has included relay state infor-
2220 mation through an additional RelayState parameter.

2221

2222 <QueryString>SAMLart=AAQAADWNEw5VT47wcO4zX%2FiEzMmFQv%3D&RelayState=0043bfc1bc45110dae17004005b13a2b</QueryString>

2223 The contents of the above element would be communicated to the invited user by the SP (either directly or indirectly
2224 through the inviting user) for them to provide to their IDP.

2225 4.4.3.3. Processing Rules

2226 To support this invitation model, when responding to either a <AddEntityRequest> or
2227 <AddKnownEntityRequest> message, the PS provider:

2228 • MUST create an artifact string in accordance with the SAML Artifact Binding (see [SAMLBind2]).

2229 • MUST insert this string within an <QueryString> element in the <AddEntityResponse> or
2230 <AddKnownEntityResponse> message.

2231 • MUST be prepared for, at some point in the future, an IDP to send an <samlp:ArtifactResolve> message as
2232 a consequence of the invited user presenting the contents of any specified <QueryString> element to the IDP.
2233 As it may be some time before the invited user does present the artifact to their IDP, the PS provider SHOULD
2234 store the artifact for a reasonable length of time.

2235 MUST return the appropriate <samlp:AuthnRequest> message in its <samlp:ArtifactResponse> message.

2236 MAY, when the IDP returns a name identifier (either pre-existing or generated) for the invited user in its
2237 <samlp:Response> message, send an <ims:IdentityMappingRequest> message to the IDP requesting an
2238 identity token (targeted at itself) for the invited user unless it already has such a identity token.

2239 After receiving an <AddEntityResponse> or <AddKnownEntityResponse> message with an <QueryString>
2240 element, the SP:

2241 • MUST extract the contents of the <QueryString> element

2242 • MUST attempt to communicate the extracted string to the invited user.

2243 If and when the invited user presents the query string that it received from the SP to its IDP, the IDP:

2244 • MUST authenticate the user

2245 • MUST determine the identity of the PS provider from the artifact string *SourceID* and determine the addresses to
2246 which <samlp:ArtifactResolve> and <Response> messages are to be sent (e.g., through metadata).

2247 • MUST send an <samlp:ArtifactResolve> message to the PS provider and MUST process the retrieved
2248 <samlp:AuthnRequest> message in accordance with the SSO Profiles of SAML [SAMLProf2].

2249 • MUST create and deliver a <samlp:Response> message to the PS provider using a SAML front-channel binding
2250 (e.g., HTTP Redirect, HTTP POST, or HTTP Artifact).

- 2251 If the value of the presented query string included any relay state information, the binding by which the <samlp:
2252 Response> message is delivered to the PS MUST support the communication of this relay state information back
2253 to the PS provider.
- 2254 Once the invited user has been redirected to the PS provider and the PS provider has obtained the <samlp:
2255 Response>, the PS provider:
- 2256 • MUST extract the name identifier from within the <Subject> of the <Assertion>.
- 2257 • MUST determine the original <AddEntityRequest> or <AddKnownEntityRequest> message to which the
2258 returned identifier corresponds.
- 2259 • MUST use the appropriate federated name identifier for the user to obtain an identity token from the IDP for the
2260 invited user - this identity token targeted for itself.
- 2261 • MUST, unless the SP did not include a <Subscription> in its <AddEntityRequest> message, obtain an identity
2262 token from the IDP for the invited user targeted at the SP.
- 2263 MUST forward on the identity token just received from the IDP in a <Notify> message, specifying the
2264 SubscriptionID of the previous <Subscription> element.
- 2265 • MUST redirect the invited user's agent to the address previously specified by the <PStoSPRedirectURL> element
2266 in the original <AddEntityRequest> or <AddKnownEntityRequest> message.
- 2267 Once the invited user has been redirected to the <PStoSPRedirectURL> address, the SP:
- 2268 • MAY choose to send a <samlp:AuthnRequest> message to the IDP asking for a <saml:AuthnStatement>
2269 attesting to that user's authentication status there. In its <Response>, the IDP MUST use the same subject identifier
2270 for this <saml:Assertion> as previously delivered to the SP within the identity token through the PS provider
2271 and the <Notify> message.

2272 5. Sequence Examples

2273 Following are detailed sequence examples for:

- 2274 • setting access control against a PS group
- 2275 • checking group membership for access control
- 2276 • performing a collective operation against group members

2277 5.1. Policy definition

2278 The following sequences demonstrates examples of the messages exchanged when a user defines access control for
2279 some SP resource in terms of group membership

2280 1. Alice visits SPa and indicates that she wishes to allow a group of hers to view some resource there.

2281 2. SPa discovers Alice's PS Provider, PSa.

2282 3. SPa queries PSa for top-level Objects.

```
2283 <ListMembersRequest Structured="children"/>
2284
2285
```

2286 4. PS responds with the Objects.

```
2287 <ListMembersResponse>
2288   <Status code="OK"/>
2289   <Object NodeType="urn:liberty:ps:entity">
2290     <ObjectID>https://psa.com/sdfhgusfsf</ObjectID>
2291     <DisplayName>Bob</DisplayName>
2292   </Object>
2293   <Object NodeType="urn:liberty:ps:entity">
2294     <ObjectID>https://psa.com/itndojd</ObjectID>
2295     <DisplayName>Mary</DisplayName>
2296   </Object>
2297   <Object NodeType="urn:liberty:ps:collection">
2298     <ObjectID>https://psa.com/sijfsfsf</ObjectID>
2299     <DisplayName>Work Friends</DisplayName>
2300   </Object>
2301   <Object NodeType="urn:liberty:ps:collection">
2302     <ObjectID>https://psa.com/lsdjfojd</ObjectID>
2303     <DisplayName>Soccer Team</DisplayName>
2304   </Object>
2305 </ListMembersResponse>
2306
2307
```

2308 5. SPs displays the list to Alice.

2309 6. Alice specifies that members of the group called **"Work Friends"** should be able to access the resource in question. |

2310 7. SPa defines appropriate permissions against the **"Work Friends"** group's ObjectID of **"https://psa.com/sijfsfsf."** |
2311 If and when somebody tries to access Alice's resource in question, at that point SPa will need to determine if that
2312 individual is a member of the group Object with this ObjectID. See the following example in [Section 5.2](#) for
2313 the sequences of messages.

2314 Rather than defining permissions against the ObjectID, the service provider could have chosen to obtain identity
2315 tokens (using a sequence of <ListMembersRequest> and <ResolveIdentifierRequest> messages) for all

2316 current members of the "Work Friends" group and then define access control rules directly against the relevant
 2317 identifiers. This may not be appropriate if the membership of the group in question is expected to change.

2318 5.2. AccessControl

2319 The following is an example of the use-case in which an SP uses group membership information for controlling access
 2320 to resources that it holds. In the use-case, Alice has defined access rules to some resources at SPa/WSCa based on
 2321 membership in a group she maintains at PSa. Bob is a friend of Alice. When Bob appears at the SPa and tries to access
 2322 the resource in question, the SPa must determine if Bob is a member of the group.

2323 1. Bob shows up at SPa and tries to access the resource in question.

2324 2. SPa asks "Who are you?"

2325 3. Bob says "Ask IDPb."

2326 4. SPa redirects Bob to IDPb with AuthnRequest.

```
2327 <samlp:AuthnRequest
2328   ID="NTT7630E00861279F0ADC63E241D0926D0B"
2329   Version="2.0" IssueInstant="...">
2330   <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2331     https://spa.com
2332   </saml:Issuer>
2333   <samlp:NameIDPolicy
2334     Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
2335 </samlp:AuthnRequest>
2336
2337
```

2338 5. IDPb authenticates Bob.

2339 6. IDPb sends a Response to SPa with an AuthnStatement carrying a name identifier for Bob.

```
2340 <samlp:Response
2341   ID="NTT3F633E3F712BAC4B0804714431D46D7B"
2342   InResponseTo="NTT7630E00861279F0ADC63E241D0926D0B"
2343   Version="2.0" IssueInstant="...">
2344   <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2345     https://idpb.com
2346   </saml:Issuer>
2347   <samlp:Status>
2348     <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
2349   </samlp:Status>
2350   <saml:Assertion
2351     Version="2.0" IssueInstant="..."
2352     ID="NTT02062BBDE3E97EF0749828BCB8C15DFB">
2353     <saml:Issuer
2354       Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2355       https://idpb.com
2356     </saml:Issuer>
2357     <saml:Subject>
2358       <saml:NameID
2359         NameQualifier="https://idpb.com"
2360         Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
2361         e0b735bf9d1f3959241d3584733d704c
2362       </saml:NameID>
2363     </saml:Subject>
2364     <saml:AuthnStatement
2365       AuthnInstant="..." SessionIndex="...">
2366       <saml:AuthnContext>AuthnContext goes here</saml:AuthnContext>
2367     </saml:AuthnStatement>
2368   </saml:Assertion>
2369
```

2370 </samlp:Response>
2371

2372 7. SPa sends IDPb an <ims:IdentityMappingRequest>, providing the previous name identifier for Bob and
2373 specifying PSa as the target namespace.

2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399

```
<soap:Envelope>
<soap:Header>
  <ws:Security>
    <saml:Assertion ID="assertionid">
      credentials for Bob at IDPb
    </saml:Assertion>
  </ws:Security>
</soap:Header>
<soap:Body>
<ims:IdentityMappingRequest>
  <ims:MappingInput>
    <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
      <samlp:NameIDPolicy SPNameQualifier="https://psa.com"
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>SPNameQualifier="https://psa.com"/>
    </sec:TokenPolicy>
    <sec:Token ref="#assertionid"/>NameQualifier="https://idpb.com"
      Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
      e0b735bf9d1f3959241d3584733d704e
    </saml:NameID>
  </sec:Token>
</ims:MappingInput>
</ims:IdentityMappingRequest>
</soap:Body>
</soap:Envelope>
```

2400 8. IDPb returns an **appropriately** mapped identifier for Bob that PSa will recognize.

2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411

```
<ims:IdentityMappingResponse>
  <ims:MappingOutput>
    <sec:Token>
      <saml:Assertion>
        identity token for Bob in PSa's namespace
      </saml:Assertion>
    </sec:Token>
  </ims:MappingOutput>
</ims:IdentityMappingResponse>
```

2412 9. As Alice has defined her access control rules in terms of a group maintained at PSa, SPa knows how to invoke
2413 PSa. SPa sends a query to PSa questioning Bob's membership in the group in question.

2414
2415
2416
2417
2418
2419
2420
2421
2422
2423

```
<TestMembershipRequest>
  <TargetObjectID>https://ps.com/sijfsfs</TargetObjectID>
  <sec:Token>
    <saml:Assertion>
      identity token for Bob in PSa's namespace
    </saml:Assertion>
  </sec:Token>
</TestMembershipRequest>
```

2424 10. PSa extracts the identity token, might decrypt the encrypted identifier in the identity token, looks up the specified
2425 group, and finds Bob's entry.

2426 11. PSa returns **"true"** to SPa.

```
2427
2428     <TestMembershipResponse>
2429     <Status code="OK"/>
2430     <TestResult>>true</TestResult>
2431 </TestMembershipResponse>
2432
```

2433 12. Confident that Bob is a member of the group against which Alice defined privileges, SPa grants Bob access to
2434 the resource in question.

2435 5.3. Group Operation

2436 The following demonstrates the sequence of steps and messages when a user desires that some operation (e.g., send
2437 an invitation) be performed on members of a particular group in their PS list.

2438 1. Alice signs on to SPa.

2439 2. Alice requests that SPa sends a party invitation to all members in a group.

2440 3. SPa/WSCa finds PSa, via DSa.

2441 4. SPa/WSCa queries PSa for the list of available groups and displays to Alice. Alice picks the relevant group on
2442 whose members she wishes to operate. SPa/WSCa requests from PSa a list of members of the specified group.

```
2443
2444     <ListMembersRequest>
2445     <TargetObjectID>https://ps.com/nmerflas</TargetObjectID>
2446 </ListMembersRequest>
2447
```

2448 5. PSa responds with a list of members to SPa/WSCa.

```
2449
2450     <ListMembersResponse>
2451     <Status code="OK"/>
2452     <Object NodeType="urn:liberty:ps:entity">
2453     <ObjectID>https://psa.com/sdfhgusfsf</ObjectID>
2454     <DisplayName>Bob</DisplayName>
2455 </Object>
2456     <Object NodeType="urn:liberty:ps:entity">
2457     <ObjectID>https://psa.com/itndojd</ObjectID>
2458     <DisplayName>Mary</DisplayName>
2459 </Object>
2460 </ListMembersResponse>
2461
```

2462 6. SPa/WSCa sends a ResolveIdentifierRequest messages with appropriate TargetObjectID elements to
2463 PSa to request identity tokens for Bob & Mary.

2464 Note: For sake of demonstration, we assume here that by chance Bob & Mary share the same IDP but this will
2465 not be the general case.

```
2466
2467     <ResolveIdentifierRequest>
2468     <ResolveInput reqID="0">
2469     <TargetObjectID>https://psa.com/sdfhgusfsf</TargetObjectID> <!-- Bob -->
2470 </ResolveInput>
2471     <ResolveInput reqID="1">
2472     <TargetObjectID>https://psa.com/itndojd</TargetObjectID> <!-- Alice -->
2473 </ResolveInput>
2474 </ResolveIdentifierRequest>
2475
```

- 2476 7. PSa sends an `ims:IdentityMappingRequest` message to IDPb including the existing identity token between
2477 PSa and IDPb, specifying SPa as the target provider.

2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503

```
<ims:IdentityMappingRequest>
  <ims:MappingInput reqID="2">
    <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion"><sec:TokenPolicy>
    <samlp:NameIDPolicy SPNameQualifier="https://spa.com"
      Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>SPNameQualifier="https://spa.com"/>
  </sec:TokenPolicy>
  <sec:Token>
    <saml:Assertion>
      existing identity token for Bob between PSa and IDPb
    </saml:Assertion>
  </sec:Token>
</ims:MappingInput>
<ims:MappingInput reqID="3">
  <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion"><sec:TokenPolicy>
  <samlp:NameIDPolicy SPNameQualifier="https://spa.com"
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>SPNameQualifier="https://spa.com"/>
</sec:TokenPolicy>
  <sec:Token>
    <saml:Assertion>
      existing identity token for Alice between PSa and IDPb
    </saml:Assertion>
  </sec:Token>
</ims:MappingInput>
</ims:IdentityMappingRequest>
```

- 2504 8. IDPb sends an `ims:IdentityMappingResponse` message with identity tokens for Bob and Mary between SPa
2505 and IDPb.

2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525

```
<ims:IdentityMappingResponse>
  <ims:MappingOutput reqRef="2">
    <Status>OK</Status>
    <sec:Token>
      <saml:Assertion>
        an identity token for Bob in SPa/WSCa's namespace goes here
      </saml:Assertion>
    </sec:Token>
  </ims:MappingOutput>
  <ims:MappingOutput reqRef="3">
    <Status>OK</Status>
    <sec:Token>
      <saml:Assertion>
        an identity token for Alice in SPa/WSCa's namespace goes here
      </saml:Assertion>
    </sec:Token>
  </ims:MappingOutput>
</ims:IdentityResponse>
```

- 2526 9. PSa forwards on the identity tokens to SPa/WSCa in its `ResolveIdentifierResponse` message to the original
2527 `ResolveIdentifierRequest` message from SPa/WSCa.

2528
2529
2530
2531
2532
2533
2534
2535
2536

```
<ResolveIdentifierResponse>
  <Status code="OK"/>
  <ResolveOutput reqRef="0">
    <Token>
      <saml:Assertion>
        an identity token for Bob in SPa/WSCa's namespace goes here
      </saml:Assertion>
    </Token>
```

```
2537     </ResolveOutput>
2538     <ResolveOutput reqRef="1">
2539       <Token>
2540         <saml:Assertion>
2541           an identity token for Alice in SPa/WSCa's namespace goes here
2542         </saml:Assertion>
2543       </Token>
2544     </ResolveOutput>
2545   </ResolveIdentifierResponse>
2546
```

- 2547 10. Once SPa/WSCa has the identity tokens for Bob & Alice, it is able to use the embedded bootstrap for Discovery
2548 Services to discover relevant WSPs, e.g., a Personal Profile service so as to get email addresses in order to send
2549 the party invitation

2550 **6. Security Considerations**

2551 A discussion of security considerations unique to the People Service and the user interaction model.

2552 • The header blocks specified in this document should be integrity-protected using the mechanisms detailed in
2553 [LibertySecMech].

2554 • Header blocks should be signed in accordance with [LibertySecMech]. The receiver of a message containing a
2555 signature that covers specific header blocks should verify the signature as part of verifying the integrity of the
2556 header block.

2557 • Metadata [LibertyMetadata] should be used to the greatest extent possible to verify message sender identity claims.

2558 • Message senders and receivers should be authenticated to one another via the mechanisms discussed in [Liberty-
2559 SecMech].

2560 **7. XML Schema for ID-WSF People Service**

2561 The formal XML schema for the ID-WSF People Service follows:

```

2562
2563 <xs:schema
2564   targetNamespace="urn:liberty:ps:2006-08"
2565   xmlns="urn:liberty:ps:2006-08"
2566   xmlns:lu="urn:liberty:util:2006-08"
2567   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2568   xmlns:ims="urn:liberty:ims:2006-08"
2569   xmlns:subs="urn:liberty:ssos:2006-08"
2570   xmlns:sec="urn:liberty:security:2006-08"
2571   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
2572   elementFormDefault="qualified"
2573   attributeFormDefault="unqualified">
2574
2575   <xs:import namespace="urn:liberty:util:2006-08"
2576     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
2577   <xs:import namespace="urn:liberty:ims:2006-08"
2578     schemaLocation="liberty-idwsf-idmapping-svc-v2.0.xsd"/>
2579   <xs:import namespace="urn:liberty:ssos:2006-08"
2580     schemaLocation="liberty-idwsf-subs-v1.0.xsd"/>
2581   <xs:import namespace="urn:liberty:security:2006-08"
2582     schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>
2583   <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
2584     schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-protocol-2.0.xsd"/>
2585
2586
2587   schemaLocation="saml-schema-protocol-2.0.xsd"/>
2588
2589   <!-->
2590   <xs:documentation>
2591     The source code in this XSD file was excerpted verbatim from:
2592
2593     Liberty ID-WSF People Service Specification
2594     Version 1.0 Draft
2595     30 July, 2006
2596
2597     Copyright (c) 2006 Liberty Alliance participants, see
2598     http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.html
2599   </xs:documentation>
2600 </xs:annotation>
2601
2602 <!-- Definition of LocalizedDisplayNameType -->
2603 <xs:complexType name="LocalizedDisplayNameType">
2604   <xs:simpleContent>
2605     <xs:extension base="xs:string">
2606       <xs:attribute name="Locale" type="xs:language" use="optional"/>
2607       <xs:attribute name="IsDefault" type="xs:boolean" use="optional"/>
2608     </xs:extension>
2609   </xs:simpleContent>
2610 </xs:complexType>
2611
2612 <!-- Definition of TagType -->
2613 <xs:complexType name="TagType">
2614   <xs:simpleContent>
2615     <xs:extension base="xs:string">
2616       <xs:attribute name="Ref" type="xs:anyURI" use="required"/>
2617     </xs:extension>
2618   </xs:simpleContent>
2619 </xs:complexType>
2620
2621 <!-- Declaration of ObjectID element -->
2622 <xs:element name="ObjectID" type="ObjectIDType"/>
2623

```

```

2624 <!-- Declaration of TargetObjectID element -->
2625 <xs:element name="TargetObjectID" type="ObjectIDType"/>
2626
2627 <!-- Definition of ObjectIDType -->
2628 <xs:complexType name="ObjectIDType">
2629   <xs:simpleContent>
2630     <xs:extension base="xs:anyURI"/>
2631   </xs:simpleContent>
2632 </xs:complexType>
2633
2634 <!-- Declaration of Object element -->
2635 <xs:element name="Object" type="ObjectType"/>
2636
2637 <!-- Definition of ObjectType -->
2638 <xs:complexType name="ObjectType">
2639   <xs:sequence>
2640     <xs:element ref="ObjectID" minOccurs="0"/>
2641     <xs:element name="DisplayName" type="LocalizedDisplayNameType"
2642       minOccurs="1" maxOccurs="unbounded"/>
2643     <xs:element name="Tag" type="TagType" minOccurs="0" maxOccurs="unbounded"/>
2644     <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2645     <xs:element name="ObjectRef" type="ObjectIDType" minOccurs="0" maxOccurs="unbounded"/>
2646   </xs:sequence>
2647   <xs:attribute name="NodeType" type="xs:anyURI" use="required"/>
2648   <xs:attribute name="CreatedDateTime" type="xs:dateTime" use="optional"/>
2649   <xs:attribute name="ModifiedDateTime" type="xs:dateTime" use="optional"/>
2650 </xs:complexType>
2651
2652 <!-- Declaration of PStoSPRedirectURL-->
2653 <xs:element name="PStoSPRedirectURL" type="PStoSPRedirectURLType"/>
2654
2655 <!-- Definition of PStoSPRedirectURLType-->
2656
2657 <xs:complexType name="PStoSPRedirectURLType">
2658   <xs:annotation>
2659     <xs:documentation>
2660       When<del> sending a AddEntityRequest to a PS provider,
2661       the SP may insert a PStoSPRedirectURL. It will be
2662       to this URL that the invited principals will be
2663       sent after federating their IDP account to the PS
2664       provider.</del>
2665     </xs:documentation>
2666     provider.</del></xs:documentation>
2667   </xs:annotation>
2668   <xs:simpleContent>
2669     <xs:extension base="xs:anyURI"/>
2670   </xs:simpleContent>
2671 </xs:complexType>
2672
2673 <!-- Declaration of SPToPSRedirectURL-->
2674 <xs:element name="SPToPSRedirectURL" type="SPToPSRedirectURLType"/>
2675
2676 <!-- Definition of SPToPSRedirectURLType-->
2677
2678 <xs:complexType name="SPToPSRedirectURLType">
2679   <xs:annotation>
2680     <xs:documentation>
2681       A<del> PS provider may insert a SPToPSRedirectURL in its
2682       AddEntityResponse. It will be to this URL that the
2683       invited principal will be sent after responding to the
2684       invitation.</del>
2685     </xs:documentation>
2686   </xs:annotation>
2687   <xs:simpleContent>
2688     <xs:extension base="xs:anyURI"/>
2689   </xs:simpleContent>
2690

```

```

2691     </xs:complexType>
2692
2693     <!-- Declaration of QueryString -->
2694
2695     <xs:element name="QueryString" type="QueryStringType"/>
2696
2697     <!-- Definition of QueryStringType-->
2698
2699     <xs:complexType name="QueryStringType">
2700         <xs:annotation>
2701             <xs:documentation>
2702                 A<xs:documentation>A PS provider may insert a QueryString in its
2703                 AddEntityResponse or AddKnownEntityResponse. The
2704                 invited Principal can present this artifact string
2705                 to a certain provider.
2706             </xs:documentation>provider.</xs:documentation>
2707         </xs:annotation>
2708         <xs:simpleContent>
2709             <xs:extension base="xs:string"/>
2710         </xs:simpleContent>
2711     </xs:complexType>
2712
2713     <!-- Declaration of CreatePSObject element -->
2714     <xs:element name="CreatePSObject"/>
2715
2716     <!-- Definition of RequestAbstractType -->
2717     <xs:complexType name="RequestAbstractType" abstract="true">
2718         <xs:anyAttribute namespace="##other" processContents="lax"/>
2719     </xs:complexType>
2720
2721     <!-- Definition of ResponseAbstractType -->
2722     <xs:complexType name="ResponseAbstractType" abstract="true">
2723         <xs:sequence>
2724             <xs:element ref="lu:Status"/>
2725         </xs:sequence>
2726         <xs:anyAttribute namespace="##other" processContents="lax"/>
2727     </xs:complexType>
2728
2729     <!-- Declaration of AddEntityRequest element -->
2730     <xs:element name="AddEntityRequest" type="AddEntityRequestType"/>
2731     <!-- Definition of AddEntityRequestType -->
2732     <xs:complexType name="AddEntityRequestType">
2733         <xs:complexContent>
2734             <xs:extension base="RequestAbstractType">
2735                 <xs:sequence>
2736                     <xs:element ref="Object"/>
2737                     <xs:element ref="PStoSPRedirectURL" minOccurs="0"/>
2738                     <xs:element ref="CreatePSObject" minOccurs="0"/>
2739                     <xs:element ref="Subscription" minOccurs="0"/>
2740                     <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
2741                 </xs:sequence>
2742             </xs:extension>
2743         </xs:complexContent>
2744     </xs:complexType>
2745
2746     <!-- Declaration of AddEntityResponse element -->
2747     <xs:element name="AddEntityResponse" type="AddEntityResponseType"/>
2748     <!-- Definition of AddEntityResponseType -->
2749     <xs:complexType name="AddEntityResponseType">
2750         <xs:complexContent>
2751             <xs:extension base="ResponseAbstractType">
2752                 <xs:sequence>
2753                     <xs:element ref="Object" minOccurs="0"/>
2754                     <xs:element ref="SPToPSRedirectURL" minOccurs="0" maxOccurs="1"/>
2755                     <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
2756                 </xs:sequence>
2757             </xs:extension>

```

```
2758     </xs:complexContent>
2759 </xs:complexType>
2760
2761 <!-- Declaration of AddKnownEntityRequest element -->
2762 <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType"/>
2763 <!-- Definition of AddKnownEntityRequestType -->
2764 <xs:complexType name="AddKnownEntityRequestType">
2765   <complexContent>
2766     <xs:extension base="RequestAbstractType">
2767       <xs:sequence>
2768         <xs:element ref="Object"/>
2769         <xs:element ref="sec:Token"/>
2770         <xs:element ref="CreatePSObject" minOccurs="0"/>
2771         <xs:element ref="Subscription" minOccurs="0"/>
2772         <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
2773       </xs:sequence>
2774     </xs:extension>
2775   </complexContent>
2776 </xs:complexType>
2777
2778 <!-- Declaration of AddKnownEntityResponse element -->
2779 <xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType"/>
2780 <!-- Definition of AddKnownEntityResponseType -->
2781 <xs:complexType name="AddKnownEntityResponseType">
2782   <complexContent>
2783     <xs:extension base="ResponseAbstractType">
2784       <xs:sequence>
2785         <xs:element ref="Object" minOccurs="0"/>
2786         <xs:element ref="SPToPSRedirectURL" minOccurs="0" maxOccurs="1"/>
2787         <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
2788       </xs:sequence>
2789     </xs:extension>
2790   </complexContent>
2791 </xs:complexType>
2792
2793 <!-- Declaration of AddCollectionRequest element -->
2794 <xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
2795 <!-- Definition of AddCollectionRequestType -->
2796 <xs:complexType name="AddCollectionRequestType">
2797   <complexContent>
2798     <xs:extension base="RequestAbstractType">
2799       <xs:sequence>
2800         <xs:element ref="Object"/>
2801         <xs:element ref="Subscription" minOccurs="0"/>
2802       </xs:sequence>
2803     </xs:extension>
2804   </complexContent>
2805 </xs:complexType>
2806
2807 <!-- Declaration of AddCollectionResponse element -->
2808 <xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>
2809 <!-- Definition of AddCollectionResponseType -->
2810 <xs:complexType name="AddCollectionResponseType">
2811   <complexContent>
2812     <xs:extension base="ResponseAbstractType">
2813       <xs:sequence>
2814         <xs:element ref="Object" minOccurs="0"/>
2815       </xs:sequence>
2816     </xs:extension>
2817   </complexContent>
2818 </xs:complexType>
2819
2820 <!-- Declaration of AddToCollectionRequest element -->
2821 <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>
2822 <!-- Definition of AddToCollectionRequestType -->
2823 <xs:complexType name="AddToCollectionRequestType">
2824   <complexContent>
```

```

2825         <xs:extension base="RequestAbstractType">
2826             <xs:sequence>
2827                 <xs:element ref="TargetObjectID"/>
2828                 <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>
2829                 <xs:element ref="Subscription" minOccurs="0"/>
2830             </xs:sequence>
2831         </xs:extension>
2832     </xs:complexContent>
2833 </xs:complexType>
2834
2835 <!-- Declaration of AddToCollectionResponse element -->
2836 <xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>
2837
2838 <!-- Declaration of RemoveEntityRequest element -->
2839 <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType"/>
2840 <!-- Definition of RemoveEntityRequestType -->
2841 <xs:complexType name="RemoveEntityRequestType">
2842     <xs:complexContent>
2843         <xs:extension base="RequestAbstractType">
2844             <xs:sequence>
2845                 <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
2846             </xs:sequence>
2847         </xs:extension>
2848     </xs:complexContent>
2849 </xs:complexType>
2850
2851 <!-- Declaration of RemoveEntityResponse element -->
2852 <xs:element name="RemoveEntityResponse" type="ResponseAbstractType"/>
2853
2854 <!-- Declaration of RemoveCollectionRequest element -->
2855 <xs:element name="RemoveCollectionRequest" type="RemoveCollectionRequestType"/>
2856 <!-- Definition of RemoveCollectionRequestType -->
2857 <xs:complexType name="RemoveCollectionRequestType">
2858     <xs:complexContent>
2859         <xs:extension base="RequestAbstractType">
2860             <xs:sequence>
2861                 <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
2862             </xs:sequence>
2863         </xs:extension>
2864     </xs:complexContent>
2865 </xs:complexType>
2866
2867 <!-- Declaration of RemoveCollectionResponse element -->
2868 <xs:element name="RemoveCollectionResponse" type="ResponseAbstractType"/>
2869
2870 <!-- Declaration of RemoveFromCollectionRequest element -->
2871 <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
2872 <!-- Definition of RemoveFromCollectionRequestType -->
2873 <xs:complexType name="RemoveFromCollectionRequestType">
2874     <xs:complexContent>
2875         <xs:extension base="RequestAbstractType">
2876             <xs:sequence>
2877                 <xs:element ref="TargetObjectID"/>
2878                 <xs:element ref="ObjectID" maxOccurs="unbounded"/>
2879                 <xs:element ref="Subscription" minOccurs="0"/>
2880             </xs:sequence>
2881         </xs:extension>
2882     </xs:complexContent>
2883 </xs:complexType>
2884
2885 <!-- Declaration of RemoveFromCollectionResponse element -->
2886 <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>
2887
2888 <!-- Declaration of ListMembersRequest element -->
2889 <xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
2890 <!-- Definition of ListMembersRequestType -->
2891 <xs:complexType name="ListMembersRequestType">

```

```
2892     <xs:complexContent>
2893       <xs:extension base="RequestAbstractType">
2894         <xs:sequence>
2895           <xs:element ref="TargetObjectID" minOccurs="0"/>
2896           <xs:element ref="Subscription" minOccurs="0"/>
2897         </xs:sequence>
2898         <xs:attribute name="Structured" type="xs:anyURI" use="optional"/>
2899         <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
2900         <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
2901       </xs:extension>
2902     </xs:complexContent>
2903 </xs:complexType>
2904
2905 <!-- Declaration of ListMembersResponse element -->
2906 <xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
2907 <!-- Definition of ListMembersResponseType -->
2908 <xs:complexType name="ListMembersResponseType">
2909   <xs:complexContent>
2910     <xs:extension base="ResponseAbstractType">
2911       <xs:sequence>
2912         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2913       </xs:sequence>
2914     </xs:extension>
2915   </xs:complexContent>
2916 </xs:complexType>
2917
2918 <!-- Declaration of QueryObjectsRequest element -->
2919 <xs:element name="QueryObjectsRequest" type="QueryObjectsRequestType"/>
2920 <!-- Definition of QueryObjectsRequestType -->
2921 <xs:complexType name="QueryObjectsRequestType">
2922   <xs:complexContent>
2923     <xs:extension base="RequestAbstractType">
2924       <xs:sequence>
2925         <xs:element name="Filter" type="xs:string"/>
2926         <xs:element ref="Subscription" minOccurs="0"/>
2927       </xs:sequence>
2928       <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
2929       <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
2930     </xs:extension>
2931   </xs:complexContent>
2932 </xs:complexType>
2933
2934 <!-- Declaration of QueryObjectsResponse element -->
2935 <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
2936 <!-- Definition of QueryObjectsResponseType -->
2937 <xs:complexType name="QueryObjectsResponseType">
2938   <xs:complexContent>
2939     <xs:extension base="ResponseAbstractType">
2940       <xs:sequence>
2941         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2942       </xs:sequence>
2943     </xs:extension>
2944   </xs:complexContent>
2945 </xs:complexType>
2946
2947 <!-- Declaration of GetObjectInfoRequest element -->
2948 <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
2949 <!-- Definition of GetObjectInfoRequestType -->
2950 <xs:complexType name="GetObjectInfoRequestType">
2951   <xs:complexContent>
2952     <xs:extension base="RequestAbstractType">
2953       <xs:sequence>
2954         <xs:element ref="TargetObjectID" minOccurs="0"/>
2955         <xs:element ref="Subscription" minOccurs="0"/>
2956       </xs:sequence>
2957     </xs:extension>
2958   </xs:complexContent>
```

```
2959     </xs:complexType>
2960
2961     <!-- Declaration of GetObjectInfoResponse element -->
2962     <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
2963     <!-- Definition of GetObjectInfoResponseType -->
2964     <xs:complexType name="GetObjectInfoResponseType">
2965         <xs:complexContent>
2966             <xs:extension base="ResponseAbstractType">
2967                 <xs:sequence>
2968                     <xs:element ref="Object" minOccurs="0"/>
2969                 </xs:sequence>
2970             </xs:extension>
2971         </xs:complexContent>
2972     </xs:complexType>
2973
2974     <!-- Declaration of SetObjectInfoRequest element -->
2975     <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
2976     <!-- Definition of SetObjectInfoRequestType -->
2977     <xs:complexType name="SetObjectInfoRequestType">
2978         <xs:complexContent>
2979             <xs:extension base="RequestAbstractType">
2980                 <xs:sequence>
2981                     <xs:element ref="Object" maxOccurs="unbounded"/>
2982                     <xs:element ref="Subscription" minOccurs="0"/>
2983                 </xs:sequence>
2984             </xs:extension>
2985         </xs:complexContent>
2986     </xs:complexType>
2987
2988     <!-- Declaration of SetObjectInfoResponse element -->
2989     <xs:element name="SetObjectInfoResponse" type="ResponseAbstractType"/>
2990
2991     <!-- Declaration of TestMembershipRequest element -->
2992     <xs:element name="TestMembershipRequest" type="TestMembershipRequestType"/>
2993     <!-- Definition of TestMembershipRequestType -->
2994     <xs:complexType name="TestMembershipRequestType">
2995         <xs:complexContent>
2996             <xs:extension base="RequestAbstractType">
2997                 <xs:sequence>
2998                     <xs:element ref="TargetObjectID" minOccurs="0"/>
2999                     <xs:element ref="sec:Token"/>
3000                     <xs:element ref="Subscription" minOccurs="0"/>
3001                 </xs:sequence>
3002             </xs:extension>
3003         </xs:complexContent>
3004     </xs:complexType>
3005
3006     <!-- Definition of ResultType -->
3007     <xs:complexType name="ResultType">
3008         <xs:simpleContent>
3009             <xs:extension base="xs:boolean"/>
3010         </xs:simpleContent>
3011     </xs:complexType>
3012
3013     <!-- Declaration of TestMembershipResponse element -->
3014     <xs:element name="TestMembershipResponse" type="TestMembershipResponseType"/>
3015     <!-- Definition of TestMembershipResponseType -->
3016     <xs:complexType name="TestMembershipResponseType">
3017         <xs:complexContent>
3018             <xs:extension base="ResponseAbstractType">
3019                 <xs:sequence>
3020                     <xs:element name="Result" type="ResultType" minOccurs="0"/>
3021                 </xs:sequence>
3022             </xs:extension>
3023         </xs:complexContent>
3024     </xs:complexType>
3025
```

```

3026 <!-- Declaration of ResolveIdentifierRequest element -->
3027 <xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
3028 <!-- Definition of ResolveIdentifierRequestType -->
3029 <xs:complexType name="ResolveIdentifierRequestType">
3030   <xs:complexContent>
3031     <xs:extension base="RequestAbstractType">
3032       <xs:sequence>
3033         <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
3034       </xs:sequence>
3035     </xs:extension>
3036   </xs:complexContent>
3037 </xs:complexType>
3038
3039 <!-- Declaration of ResolveInput element -->
3040 <xs:element name="ResolveInput" type="ResolveInputType"/>
3041 <!-- Definition of ResolveInputType -->
3042 <xs:complexType name="ResolveInputType">
3043   <xs:complexContent>
3044     <xs:extension base="ims:MappingInputType">
3045       <xs:sequence>
3046         <xs:element ref="TargetObjectID" minOccurs="0"/>
3047       </xs:sequence>
3048     </xs:extension>
3049   </xs:complexContent>
3050 </xs:complexType>
3051
3052 <!-- Declaration of ResolveIdentifierResponse element -->
3053 <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType"/>
3054 <!-- Definition of ResolveIdentifierResponseType -->
3055 <xs:complexType name="ResolveIdentifierResponseType">
3056   <xs:complexContent>
3057     <xs:extension base="ResponseAbstractType">
3058       <xs:sequence>
3059         <xs:element ref="ResolveOutput" maxOccurs="unbounded"/>
3060       </xs:sequence>
3061     </xs:extension>
3062   </xs:complexContent>
3063 </xs:complexType>
3064
3065 <!-- Declaration of ResolveOutput element -->
3066 <xs:element name="ResolveOutput" type="ims:MappingOutputType"/>
3067
3068 <!-- Declaration of Subscription element -->
3069 <xs:element name="Subscription" type="subs:SubscriptionType"/>
3070
3071 <!-- Declaration of Notification element -->
3072 <xs:element name="Notification" type="NotificationType"/>
3073 <!-- Definition of NotificationType -->
3074 <xs:complexType name="NotificationType">
3075   <xs:complexContent>
3076     <xs:extension base="subs:NotificationType">
3077       <xs:sequence>
3078         <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded"/>
3079       </xs:sequence>
3080     </xs:extension>
3081   </xs:complexContent>
3082 </xs:complexType>
3083
3084 <!-- Declaration of ItemData element -->
3085 <xs:element name="ItemData" type="ItemDataType"/>
3086 <!-- Definition of ItemDataType -->
3087 <xs:complexType name="ItemDataType">
3088   <xs:choice>
3089     <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
3090     <xs:element ref="sec:Token" minOccurs="0"/><del ref="Object"/>
3091   </xs:choice>
3092 </xs:complexType>

```

```
3093
3094 <!-- Declaration of Notify element -->
3095 <xs:element name="Notify" type="NotifyType"/>
3096 <!-- Definition of NotifyType -->
3097 <xs:complexType name="NotifyType">
3098   <xs:complexContent>
3099     <xs:extension base="RequestAbstractType">
3100       <xs:sequence>
3101         <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
3102       </xs:sequence>
3103       <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
3104     </xs:extension>
3105   </xs:complexContent>
3106 </xs:complexType>
3107
3108 <!-- Declaration of NotifyResponse element -->
3109 <xs:element name="NotifyResponse" type="subs:NotifyResponseType"/>
3110
3111 </xs:schema>
3112
3113
```

3114 **8. Abstract WSDL**

```

3115
3116 <definitions
3117     name="id-wsf-ps_2006-08_wsd1_interface"
3118     targetNamespace="urn:liberty:ps:2006-08"
3119     xmlns:tns="urn:liberty:ps:2006-08"
3120     xmlns="http://schemas.xmlsoap.org/wsd1/"
3121     xmlns:soap="http://schemas.xmlsoap.org/wsd1/soap/"
3122     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3123     xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsd1"
3124     xmlns:ps="urn:liberty:ps:2006-08"
3125     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3126     xsi:schemaLocation="http://schemas.xmlsoap.org/wsd1/
3127         http://schemas.xmlsoap.org/wsd1/
3128         http://www.w3.org/2006/02/addressing/wsd1
3129         http://www.w3.org/2006/02/addressing/wsd1/ws-addr-wsd1.xsd">
3130
3131     <xsd:documentation>
3132
3133     XML Schema from Liberty People Service Specification.
3134
3135     ### NOTICE ###
3136
3137     Copyright (c) 2006 Liberty Alliance participants, see
3138     http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
3139
3140     </xsd:documentation>
3141
3142     <types>
3143         <xsd:schema>
3144             <xsd:import namespace="urn:liberty:ps:2006-08"
3145                 schemaLocation="liberty-idwsf-people-service-v1.0.xsd"/>
3146         </xsd:schema>
3147     </types>
3148
3149     <!-- Messages -->
3150
3151     <!-- Adding a User -->
3152
3153     <message name="AddEntityRequest">
3154         <part name="body" element="ps:AddEntityRequest"/>
3155     </message>
3156
3157     <message name="AddEntityResponse">
3158         <part name="body" element="ps:AddEntityResponse"/>
3159     </message>
3160
3161     <!-- Adding a Known User -->
3162
3163     <message name="AddKnownEntityRequest">
3164         <part name="body" element="ps:AddKnownEntityRequest"/>
3165     </message>
3166
3167     <message name="AddKnownEntityResponse">
3168         <part name="body" element="ps:AddKnownEntityResponse"/>
3169     </message>
3170
3171     <!-- Removing a User -->
3172
3173     <message name="RemoveEntityRequest">
3174         <part name="body" element="ps:RemoveEntityRequest"/>
3175     </message>
3176
3177     <message name="RemoveEntityResponse">

```

```
3179     <part name="body" element="ps:RemoveEntityResponse" />
3180 </message>
3181
3182 <!-- Adding a Group -->
3183
3184 <message name="AddCollectionRequest">
3185     <part name="body" element="ps:AddCollectionRequest" />
3186 </message>
3187
3188 <message name="AddCollectionResponse">
3189     <part name="body" element="ps:AddCollectionResponse" />
3190 </message>
3191
3192 <!-- Removing a Group -->
3193
3194 <message name="RemoveCollectionRequest">
3195     <part name="body" element="ps:RemoveCollectionRequest" />
3196 </message>
3197
3198 <message name="RemoveCollectionResponse">
3199     <part name="body" element="ps:RemoveCollectionResponse" />
3200 </message>
3201
3202 <!-- Adding to a Group -->
3203
3204 <message name="AddToCollectionRequest">
3205     <part name="body" element="ps:AddToCollectionRequest" />
3206 </message>
3207
3208 <message name="AddToCollectionResponse">
3209     <part name="body" element="ps:AddToCollectionResponse" />
3210 </message>
3211
3212 <!-- Removing From a Group -->
3213
3214 <message name="RemoveFromCollectionRequest">
3215     <part name="body" element="ps:RemoveFromCollectionRequest" />
3216 </message>
3217
3218 <message name="RemoveFromCollectionResponse">
3219     <part name="body" element="ps:RemoveFromCollectionResponse" />
3220 </message>
3221
3222 <!-- Listing Members -->
3223
3224 <message name="ListMembersRequest">
3225     <part name="body" element="ps:ListMembersRequest" />
3226 </message>
3227
3228 <message name="ListMembersResponse">
3229     <part name="body" element="ps:ListMembersResponse" />
3230 </message>
3231
3232 <!-- Retrieving Object Info -->
3233
3234 <message name="GetObjectInfoRequest">
3235     <part name="body" element="ps:GetObjectInfoRequest" />
3236 </message>
3237
3238 <message name="GetObjectInfoResponse">
3239     <part name="body" element="ps:GetObjectInfoResponse" />
3240 </message>
3241
3242 <!-- Updating Object Info -->
3243
3244 <message name="SetObjectInfoRequest">
3245     <part name="body" element="ps:SetObjectInfoRequest" />
```

```

3246     </message>
3247
3248     <message name="SetObjectInfoResponse">
3249         <part name="body" element="ps:SetObjectInfoResponse"/>
3250     </message>
3251
3252     <!-- Querying Objects -->
3253
3254     <message name="QueryObjectsRequest">
3255         <part name="body" element="ps:QueryObjectsRequest"/>
3256     </message>
3257
3258     <message name="QueryObjectsResponse">
3259         <part name="body" element="ps:QueryObjectsResponse"/>
3260     </message>
3261
3262     <!-- Testing Membership -->
3263
3264     <message name="TestMembershipRequest">
3265         <part name="body" element="ps:TestMembershipRequest"/>
3266     </message>
3267
3268     <message name="TestMembershipResponse">
3269         <part name="body" element="ps:TestMembershipResponse"/>
3270     </message>
3271
3272     <!-- Resolving Identifiers -->
3273
3274     <message name="ResolveIdentifierRequest">
3275         <part name="body" element="ps:ResolveIdentifierRequest"/>
3276     </message>
3277
3278     <message name="ResolveIdentifierResponse">
3279         <part name="body" element="ps:ResolveIdentifierResponse"/>
3280     </message>
3281
3282     <!-- Port Type -->
3283
3284     <portType name="LibertyPS1">
3285
3286         <operation name="AddEntity">
3287             <input message="tns:AddEntityRequest"
3288                 wsaw:Action="urn:liberty:ps:2006-08:AddEntityRequest"/>
3289             <output message="tns:AddEntityResponse"
3290                 wsaw:Action="urn:liberty:ps:2006-08:AddEntityResponse"/>
3291             <input message="tns:AddEntityRequest"
3292                 wsaw:Action="urn:liberty:ps:2006-08:AddEntityRequest"/>
3293             <output message="tns:AddEntityResponse"
3294                 wsaw:Action="urn:liberty:ps:2006-08:AddEntityResponse"/>
3295         </operation>
3296
3297         <operation name="AddKnownEntity">
3298             <input message="tns:AddKnownEntityRequest"
3299                 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityRequest"/>
3300             <output message="tns:AddKnownEntityResponse"
3301                 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityResponse"/>
3302             <input message="tns:AddKnownEntityRequest"
3303                 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityRequest"/>
3304             <output message="tns:AddKnownEntityResponse"
3305                 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityResponse"/>
3306         </operation>
3307
3308         <operation name="RemoveEntity">
3309             <input message="tns:RemoveEntityRequest"
3310                 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityRequest"/>
3311             <output message="tns:RemoveEntityResponse"
3312                 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityResponse"/>
3313             <input message="tns:RemoveEntityRequest"
3314                 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityRequest"/>
3315             <output message="tns:RemoveEntityResponse"
3316                 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityResponse"/>
3317         </operation>

```

```
3313     <operation name="AddCollection">
3314         <input message="tns:AddCollectionRequest"
3315             wsaw:Action="urn:liberty:ps:2006-08:AddCollectionRequest"/>
3316         wsaw:Action="urn:liberty:ps:2006-08:AddCollectionRequest" />
3317     <output message="tns:AddCollectionResponse"
3318         wsaw:Action="urn:liberty:ps:2006-08:AddCollectionResponse"/>
3319     wsaw:Action="urn:liberty:ps:2006-08:AddCollectionResponse" />
3320 </operation>
3321
3322     <operation name="RemoveCollection">
3323         <input message="tns:RemoveCollectionRequest"
3324             wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionRequest"/>
3325         wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionRequest" />
3326     <output message="tns:RemoveCollectionResponse"
3327         wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionResponse"/>
3328     wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionResponse" />
3329 </operation>
3330
3331     <operation name="AddToCollection">
3332         <input message="tns:AddToCollectionRequest"
3333             wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionRequest"/>
3334         wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionRequest" />
3335     <output message="tns:AddToCollectionResponse"
3336         wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionResponse"/>
3337     wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionResponse" />
3338 </operation>
3339
3340     <operation name="RemoveFromCollection">
3341         <input message="tns:RemoveFromCollectionRequest"
3342             wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionRequest"/>
3343         wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionRequest" />
3344     <output message="tns:RemoveFromCollectionResponse"
3345         wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionResponse"/>
3346     wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionResponse" />
3347 </operation>
3348
3349     <operation name="ListMembersOfCollection">
3350         <input message="tns:ListMembersRequest"
3351             wsaw:Action="urn:liberty:ps:2006-08:ListMembersRequest"/>
3352         wsaw:Action="urn:liberty:ps:2006-08:ListMembersRequest" />
3353     <output message="tns:ListMembersResponse"
3354         wsaw:Action="urn:liberty:ps:2006-08:ListMembersResponse"/>
3355     wsaw:Action="urn:liberty:ps:2006-08:ListMembersResponse" />
3356 </operation>
3357
3358     <operation name="GetObjectInfo">
3359         <input message="tns:GetObjectInfoRequest"
3360             wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoRequest"/>
3361         wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoRequest" />
3362     <output message="tns:GetObjectInfoResponse"
3363         wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoResponse"/>
3364     wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoResponse" />
3365 </operation>
3366
3367     <operation name="SetObjectInfo">
3368         <input message="tns:SetObjectInfoRequest"
3369             wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoRequest"/>
3370         wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoRequest" />
3371     <output message="tns:SetObjectInfoResponse"
3372         wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoResponse"/>
3373     wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoResponse" />
3374 </operation>
3375
3376     <operation name="QueryObjects">
3377         <input message="tns:QueryObjectsRequest"
3378             wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsRequest"/>
3379         wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsRequest" />
```

```

3380 -----<output message="tns:QueryObjectsResponse"
3381         wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsResponse"/>
3382     wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsResponse" />
3383 -----</operation>
3384
3385     <operation name="TestMembership">
3386         <input message="tns:TestMembershipRequest"
3387             wsaw:Action="urn:liberty:ps:2006-08:TestMembershipRequest"/>
3388         wsaw:Action="urn:liberty:ps:2006-08:TestMembershipRequest" />
3389 -----<output message="tns:TestMembershipResponse"
3390         wsaw:Action="urn:liberty:ps:2006-08:TestMembershipResponse"/>
3391     wsaw:Action="urn:liberty:ps:2006-08:TestMembershipResponse" />
3392 -----</operation>
3393
3394     <operation name="ResolveIdentifier">
3395         <input message="tns:ResolveIdentifierRequest"
3396             wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierRequest"/>
3397         wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierRequest" />
3398 -----<output message="tns:ResolveIdentifierResponse"
3399         wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierResponse"/>
3400     wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierResponse" />
3401 -----</operation>
3402
3403 </portType>
3404
3405 <!--
3406 An example of a binding and service that can be used with this
3407 abstract service description is provided below.
3408 --->
3409
3410     <binding name="PeopleServiceSoapBinding" type="tns:LibertyPS1">
3411         <soap:binding style="document"
3412             transport="http://schemas.xmlsoap.org/soap/http"/>
3413         <transport="http://schemas.xmlsoap.org/soap/http" />
3414
3415         <operation name="AddEntity">
3416             <soap:operation />
3417             <soap:operation soapAction="urn:liberty:ps:2006-08:AddEntityRequest" />
3418 -----<input> <soap:body use="literal"/>use="literal" /></input>
3419             <output> <soap:body use="literal" /> /></output>
3420         </operation>
3421
3422         <operation name="AddKnownEntity">
3423             <soap:operation soapAction="urn:liberty:ps:2006-08:AddKnownEntityRequest" />
3424 -----<input> <soap:body use="literal" /> /></input>
3425             <output> <soap:body use="literal" />use="literal" /></output>
3426         </operation>
3427
3428         <operation name="RemoveEntity">
3429             <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveEntityRequest" />
3430 -----<input> <soap:body use="literal" /> /></input>
3431             <output> <soap:body use="literal" /> /></output>
3432         </operation>
3433
3434         <operation name="AddCollection">
3435             <soap:operation soapAction="urn:liberty:ps:2006-08:AddCollectionRequest" />
3436 -----<input> <soap:body use="literal" />use="literal" /></input>
3437             <output> <soap:body use="literal" /> /></output>
3438         </operation>
3439
3440         <operation name="RemoveCollection">
3441             <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveCollectionRequest" />
3442 -----<input> <soap:body use="literal" />use="literal" /></input>
3443             <output> <soap:body use="literal" /> /></output>
3444         </operation>
3445
3446         <operation name="RemoveCollection">
3447             <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveCollectionRequest" />

```

Liberty ID-WSF People Service Specification

```

3447 <input> <soap:body use="literal"/> /></input>
3448 <output> <soap:body use="literal"/>use="literal"/></output>
3449 </operation>
3450 <operation name="AddToCollection">
3451
3452 <soap:operation soapAction="urn:liberty:ps:2006-08:AddToCollectionRequest"/>
3453 <input> <soap:body use="literal"/> /></input>
3454 <output> <soap:body use="literal"/>use="literal"/></output>
3455 </operation>
3456 <operation name="RemoveFromCollection">
3457
3458 <soap:operation soapAction="urn:liberty:ps:2006-08:RemoveFromCollectionRequest"/>
3459 <input> <soap:body use="literal"/> /></input>
3460 <output> <soap:body use="literal"/> /></output>
3461 </operation>
3462 <operation name="ListMembersOfCollection">
3463
3464 <soap:operation soapAction="urn:liberty:ps:2006-08:ListMembersOfCollectionRequest"/>
3465 <input> <soap:body use="literal"/> /></input>
3466 <output> <soap:body use="literal"/>use="literal"/></output>
3467 </operation>
3468 <operation name="GetObjectInfo">
3469
3470 <soap:operation soapAction="urn:liberty:ps:2006-08:GetObjectInfoRequest"/>
3471 <input> <soap:body use="literal"/> /></input>
3472 <output> <soap:body use="literal"/>use="literal"/></output>
3473 </operation>
3474 <operation name="SetObjectInfo">
3475
3476 <soap:operation soapAction="urn:liberty:ps:2006-08:SetObjectInfoRequest"/>
3477 <input> <soap:body use="literal"/> /></input>
3478 <output> <soap:body use="literal"/>use="literal"/></output>
3479 </operation>
3480 <operation name="QueryObjects">
3481
3482 <soap:operation soapAction="urn:liberty:ps:2006-08:QueryObjectsRequest"/>
3483 <input> <soap:body use="literal"/> /></input>
3484 <output> <soap:body use="literal"/>use="literal"/></output>
3485 </operation>
3486 <operation name="TestMembership">
3487
3488 <soap:operation soapAction="urn:liberty:ps:2006-08:TestMembershipRequest"/>
3489 <input> <soap:body use="literal"/> /></input>
3490 <output> <soap:body use="literal"/>use="literal"/></output>
3491 </operation>
3492 <operation name="ResolveIdentifier">
3493
3494 <soap:operation soapAction="urn:liberty:ps:2006-08:ResolveIdentifierRequest"/>
3495 <input> <soap:body use="literal"/>use="literal"/></input>
3496 <output> <soap:body use="literal"/> /></output>
3497 </operation>
3498
3499 </binding>
3500
3501 <service name="PeopleService">
3502 <port name="PeoplePort" binding="ps:PeopleServiceSoapBinding">
3503
3504 <!-- Modify with the REAL SOAP endpoint -->
3505
3506 <soap:address location="http://example.com/peopleservice"/>
3507 location="http://example.com/peopleservice"/>
3508 </port>
3509 </service>
3510
3511 </definitions>
3512
3513

```

3514 References

3515 Normative

- 3516 [LibertyAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication, Single
3517 Sign-On, and Identity Mapping Services Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (28
3518 November, 2006). <http://www.projectliberty.org/specs>
- 3519 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-
3520 errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>
- 3521 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
3522 2006). <http://www.projectliberty.org/specs>
- 3523 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
3524 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>
- 3525 [LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0 errata v1.0, Liberty Alliance Project (21 April, 2007).
3526 <http://www.projectliberty.org/specs>
- 3527 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0,
3528 Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>
- 3529 [LibertySUBS] Kellomäki, Sampo, eds. "Liberty ID-WSF Subscriptions and Notifications," Version 1.0, Liberty Al-
3530 liance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 3531 [LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata," Ver-
3532 sion 1.0, Liberty Alliance Project (13 April, 2007). <http://www.projectliberty.org/specs>
- 3533 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet Engi-
3534 neering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 3535 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions and Protocol
3536 for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organ-
3537 ization for the Advancement of Structured Information Standards [http://docs.oasis-open.org/security/saml/](http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
3538 [v2.0/saml-core-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 3539 [SAMLBind2] Cantor, Scott, Hirsch, Frederick, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Bind-
3540 ings for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard,
3541 Organization for the Advancement of Structured Information Standards [http://docs.oasis-open.org/security/](http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf)
3542 [saml/v2.0/saml-bindings-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf)
- 3543 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
3544 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
3545 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards [http://](http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf)
3546 docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf
- 3547 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.
3548 World Wide Web Consortium W3C Recommendation (9 May 2006). [http://www.w3.org/TR/2006/REC-ws-](http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/)
3549 [addr-core-20060509/](http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/)
- 3550 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
3551 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consorti-
3552 um <http://www.w3.org/TR/2004/REC-xml-20040204>

- 3553 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October 2004).
3554 "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium [http://](http://www.w3.org/TR/xmlschema-1/)
3555 www.w3.org/TR/xmlschema-1/