



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-errata-v1.0.pdf

1 **Notice**

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

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

15 Copyright © 2007 2FA Technology; Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.;
16 American Express Company; Amssoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Bank of
17 America Corporation; Beta Systems Software AG; 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
20 Systems LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
21 développement de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens Ltd.; GSA Office of
22 Governmentwide Policy; Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke & Devrient GmbH;
23 Hewlett-Packard Company; Hochhauser & Co., LLC; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega;
24 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
27 Corporation; Novell, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation; RSA Security Inc.;
28 Reactivity Inc.; Royal Mail Group plc; SAP AG; Senforce; Sharp Laboratories of America; Sigaba; SmartTrust; Sony
29 Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.; Telecom Italia S.p.A.;
30 Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security; Trusted Network Technologies; UNINETT AS; UTI;
31 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

Contents

38		
39	1. Introduction	6
40	1.1. Overview	6
41	1.2. Notation	6
42	1.3. Terminology	7
43	1.4. Namespaces	7
44	2. Data Model	8
45	2.1. <Object> Element	8
46	2.1.1. NodeType Attribute	8
47	2.1.2. CreatedDateTime Attribute	8
48	2.1.3. ModifiedDateTime Attribute	8
49	2.1.4. <ObjectID> Element	9
50	2.1.5. <DisplayName> Element	9
51	2.1.6. <Tag> Element	9
52	2.1.7. <ObjectRef> Element	10
53	2.2. <Token> Element	10
54	3. People Service	11
55	3.1. Overview	11
56	3.2. Service Type	11
57	3.3. Action URIs	11
58	3.4. Request and Response Abstract Types	12
59	3.4.1. Complex Type RequestAbstractType	12
60	3.4.2. Complex Type ResponseAbstractType	12
61	3.5. Status	12
62	3.6. Identity Token Policy	14
63	3.7. Success & Failure	14
64	3.8. Subscription and Notification	14
65	3.8.1. <Subscription> Element	15
66	3.8.2. Notify and NotifyResponse Messages	16
67	3.8.3. <Notification> Element	16
68	3.9. Adding an Entity	17
69	3.9.1. wsa:Action Values	17
70	3.9.2. AddEntityRequest Message	17
71	3.9.3. AddEntityResponse Message	18
72	3.9.4. Processing Rules	19
73	3.10. Adding a Known Entity	21
74	3.10.1. wsa:Action Values	21
75	3.10.2. AddKnownEntityRequest Message	21
76	3.10.3. AddKnownEntityResponse Message	23
77	3.10.4. Processing Rules	24
78	3.11. Removing an Entity	25
79	3.11.1. wsa:Action Values	25
80	3.11.2. RemoveEntityRequest Message	25
81	3.11.3. RemoveEntityResponse Message	26
82	3.11.4. Processing Rules	26
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	27
87	3.12.4. Processing Rules	28
88	3.13. Removing a Collection	28
89	3.13.1. wsa:Action Values	28
90	3.13.2. RemoveCollectionRequest Message	29

91	3.13.3. RemoveCollectionResponse Message	29
92	3.13.4. Processing Rules	30
93	3.14. Adding to a Collection	30
94	3.14.1. wsa:Action Values	30
95	3.14.2. AddToCollectionRequest Message	30
96	3.14.3. AddToCollectionResponse Message	31
97	3.14.4. Processing Rules	32
98	3.15. Removing from a Collection	32
99	3.15.1. wsa:Action Values	32
100	3.15.2. RemoveFromCollectionRequest Message	32
101	3.15.3. RemoveFromCollectionResponse Message	33
102	3.15.4. Processing Rules	34
103	3.16. Listing Members	34
104	3.16.1. wsa:Action Values	34
105	3.16.2. ListMembersRequest Message	34
106	3.16.3. ListMembersResponse Message	36
107	3.16.4. Examples	36
108	3.16.5. Processing Rules	38
109	3.17. Retrieving Info	39
110	3.17.1. wsa:Action Values	40
111	3.17.2. GetObjectInfoRequest Message	40
112	3.17.3. GetObjectInfoResponse Message	40
113	3.17.4. Processing Rules	41
114	3.18. Updating Info	41
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	43
120	3.19.1. wsa:Action Values	43
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	49
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. <SpttoSPRedirectURL> element	54
141	4.4.3. <QueryString> element	55
142	5. Sequence Examples	58
143	5.1. Policy definition	58
144	5.2. AccessControl	58

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	79

1. Introduction

1.1. Overview

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

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

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

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

There are two significant implications of this model:

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

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

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

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

1.2. Notation

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

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "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
196 capitalized, 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
203 provides online services to users through HTTP interactions. In interactions with other providers not mediated by a
204 user's browser, websites assume the role of a WSC in order to send SOAP-based requests. For clarity, this specification
205 uses 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:2006-08`).
209 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`)
211 [Schema1-2].
- 212 • The prefix `xml:` stands for the W3C XML namespace (`http://www.w3.org/XML/1998/namespace`) [XML].
- 213 • The prefix `saml:` stands for the OASIS SSTC SAML2.0 Assertion namespace (`urn:oasis:names:tc:SAML:2.0:assertion`)
214 [SAMLCore2].
- 215 • The prefix `samlp:` stands for the OASIS SSTC SAML2.0 Protocol namespace (`urn:oasis:names:tc:SAML:2.0:protocol`)
216 [SAMLCore2].
- 217 • The prefix `ims:` stands for the Liberty ID-WSF Authentication Service Identity Mapping Service namespace
218 (`urn:liberty:ims:2006-08`) [LibertyAuthn].
- 219 • The prefix `sec:` stands for the Liberty ID-WSF Security Mechanisms Core namespace (`urn:liberty:sec:2005-11`)
220 [LibertySecMech].
- 221 • The prefix `subs:` stands for the Liberty ID-WSF Subscriptions & Notifications namespace
222 (`urn:liberty:ssos:2006-08`) [LibertySUBS].

2. Data Model

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

2.1. <Object> Element

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

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

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

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

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

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

```
<xs:element name="Object" type="ObjectType" />
<xs:complexType name="ObjectType">
  <xs:sequence>
    <xs:element ref="ObjectID" minOccurs="0" />
    <xs:element name="DisplayName" type="LocalizedDisplayNameType"
      minOccurs="1" maxOccurs="unbounded" />
    <xs:element name="Tag" type="TagType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="ObjectRef" type="ObjectIDType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="NodeType" type="xs:anyURI" use="required" />
  <xs:attribute name="CreatedDateTime" type="xs:dateTime" use="optional" />
  <xs:attribute name="ModifiedDateTime" type="xs:dateTime" use="optional" />
</xs:complexType>
```

2.1.1. NodeType Attribute

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

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

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

2.1.2. CreatedDateTime Attribute

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

2.1.3. ModifiedDateTime Attribute

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

2.1.4. <ObjectID> Element

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

```
<!-- Declaration of ObjectID element -->
<xs:element name="ObjectID" type="ObjectIDType" />
<!-- Definition of ObjectIDType -->
<xs:complexType name="ObjectIDType">
  <xs:simpleContent>
    <xs:restriction base="xs:anyURI" />
  </xs:simpleContent>
</xs:complexType>
```

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

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

2.1.5. <DisplayName> Element

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

```
<xs:complexType name="LocalizedDisplayNameType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="Locale" type="xs:language" use="optional" />
      <xs:attribute name="IsDefault" type="boolean" use="optional" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

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

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

2.1.6. <Tag> Element

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

```
313  
314 <xs:complexType name="TagType">  
315   <xs:attribute name="Ref" type="xs:anyURI" use="required"/>  
316 </xs:complexType>  
317
```

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

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

322 **2.1.7. <ObjectRef> Element**

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

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

327 **2.2. <Token> Element**

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

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

3. People Service

3.1. Overview

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

3.2. Service Type

A People Service is identified by the service type URN:

urn:liberty:ps:2006-08

3.3. Action URIs

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

This specification defines the following action identifiers:

- *urn:liberty:ps:2006-08:AddEntityRequest*
- *urn:liberty:ps:2006-08:AddEntityResponse*
- *urn:liberty:ps:2006-08:AddKnownEntityRequest*
- *urn:liberty:ps:2006-08:AddKnownEntityResponse*
- *urn:liberty:ps:2006-08:RemoveEntityRequest*
- *urn:liberty:ps:2006-08:RemoveEntityResponse*
- *urn:liberty:ps:2006-08:AddCollectionRequest*
- *urn:liberty:ps:2006-08:AddCollectionResponse*
- *urn:liberty:ps:2006-08:RemoveCollectionRequest*
- *urn:liberty:ps:2006-08:RemoveCollectionResponse*
- *urn:liberty:ps:2006-08:AddToCollectionRequest*
- *urn:liberty:ps:2006-08:AddToCollectionResponse*
- *urn:liberty:ps:2006-08:RemoveFromCollectionRequest*
- *urn:liberty:ps:2006-08:RemoveFromCollectionResponse*
- *urn:liberty:ps:2006-08:ListMembersRequest*
- *urn:liberty:ps:2006-08:ListMembersResponse*
- *urn:liberty:ps:2006-08:GetObjectInfoRequest*

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

373 **3.4. Request and Response Abstract Types**

374 **3.4.1. Complex Type RequestAbstractType**

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

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

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

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

384 **3.4.2. Complex Type ResponseAbstractType**

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

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

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

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

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

```
392
393 <!-- Definition of ResponseAbstractType -->
394 <xs:complexType name="ResponseAbstractType" abstract="true">
395   <xs:sequence>
396     <xs:element ref="lu:Status"/>
397   </xs:sequence>
398   <xs:anyAttribute namespace="##other" processContents="lax"/>
399 </xs:complexType>
400
```

401 **3.5. Status**

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

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

- 427 • Timeout
- 428 • UnexpectedError
- 429 • UnrecognizedFilter
- 430 • UnrecognizedNamespace
- 431 • UnspecifiedError

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

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

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

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

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

447 3.6. Identity Token Policy

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

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

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

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

459 3.7. Success & Failure

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

463 3.8. Subscription and Notification

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

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

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

471 **3.8.1. <Subscription> Element**

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

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

```
477  
478 <xs:element name="Subscription" type="subs:SubscriptionType" />  
479
```

480 **3.8.1.1. Selecting Objects**

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

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

485 **3.8.1.2. Triggers**

486 This specification defines no triggers.

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

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

492 **3.8.1.3. Subscription Start**

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

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

496 **3.8.1.4. Subscription Aggregation**

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

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

499 **3.8.1.5. Subscription Expiration**

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

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

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

506 3.8.1.6. Subscription Querying and Management

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

509 3.8.1.7. Including Data

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

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

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

515 3.8.2. Notify and NotifyResponse Messages

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

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

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

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

538 3.8.3. <Notification> Element

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

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

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

```
546 <xs:element name="Notification" type="NotificationType" />
547
548 <xs:complexType name="NotificationType">
549   <xs:complexContent>
550     <xs:extension base="subs:NotificationType">
551       <xs:sequence>
552         <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded" />
553       </xs:sequence>
554     </xs:extension>
555   </xs:complexContent>
556 </xs:complexType>
557
558
```

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

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

570 3.9. Adding an Entity

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

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

574 3.9.1. wsa:Action Values

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

578 3.9.2. AddEntityRequest Message

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

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

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

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

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

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

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

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

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

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

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

```
604
605 <!-- Declaration of AddEntityRequest element -->
606 <xs:element name="AddEntityRequest" type="AddEntityRequestType"/>
607 <!-- Definition of AddEntityRequestType -->
608 <xs:complexType name="AddEntityRequestType">
609   <xs:complexContent>
610     <xs:extension base="RequestAbstractType">
611       <xs:sequence>
612         <xs:element ref="Object"/>
613         <xs:element ref="PStoSPRedirectURL" minOccurs="0"/>
614         <xs:element ref="CreatePSObject" minOccurs="0"/>
615         <xs:element ref="Subscription" minOccurs="0"/>
616         <xs:element ref="TokenPolicy" minOccurs="0"/>
617       </xs:sequence>
618     </xs:extension>
619   </xs:complexContent>
620 </xs:complexType>
621
```

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

```
626
627 <AddEntityRequest>
628   <Object NodeType="urn:liberty:ps:entity">
629     <DisplayName>Alison</DisplayName>
630   </Object>
631   <PStoSPRedirectURL>some SP URL</PStoSPRedirectURL>
632 </AddEntityRequest>
633
```

634 3.9.3. AddEntityResponse Message

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

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

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

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

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

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

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

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

```
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 </xs:complexType>
672
```

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

```
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

The WSC:

- MUST include an `<Object>` element within the `<AddEntityRequest>` message.
- MUST include a `NodeType` attribute on the `<Object>` element with a value of `urn:liberty:ps:entity`.
- MUST include at least one `<DisplayName>` element for the invited user within the `<Object>` element. This element contains the friendly name that the user desires be used for the created `urn:liberty:ps:entity` Object.
- MAY include a `<PStoSPRedirectURL>` element.
- MAY, if it desires that the PS provider create (if not already existing) an `Object` for the inviting user at the PS provider of the invited user, include a `<CreatePSObject>` element.
- MAY, if it desires that an identity token be returned to it through a subsequent `<Notification>` message, include a `<Subscription>` element. The presence of a `<Subscription>` element indicates to the PS provider that the SP desires that the PS provider return to it (when later possible) an identity token for the invited user within a `<Notification>` in a `<Notify>` message - this is possible after a federation has been established between the PS provider and the appropriate IDP. If no `<Subscription>` element is present, the SP is indicating that the PS provider need not return an identity token through this mechanism.
If the WSC includes a `<Subscription>` element, it MAY specify its requirements of the eventually returned identity token by including a `<sec:TokenPolicy>` element. The WSC SHOULD NOT include a `<sec:TokenPolicy>` element unless also including a `<Subscription>` element.

In responding to a successful `<AddEntityRequest>` message, the PS provider:

- MUST create a new `Object` element as a direct child of the root node.
- MUST include an `ObjectID` in the returned `Object`.
- SHOULD include either or both of a `<SPtoPSRedirectURL>` or `<QueryString>` element in the `<AddEntityResponse>` message returned to the calling SP.
- MUST be prepared for the invited user to, at some point in the future, visit the URL provided in any specified `<SPtoPSRedirectURL>` element. As it may be some time before the invited user does respond, the PS provider SHOULD store such a URL for a reasonable length of time.
MUST, if and when the invited user does respond to the URL specified by the `<SPtoPSRedirectURL>` element, endeavor to establish a federated identifier for that user with the appropriate identity provider (see [Section 4](#))
SHOULD, if and when such a federated identifier is established, send an `<ims:IdentityMappingRequest>` message to that IDP requesting a long-lived identity token (targeted at itself as the provider) for the user for which the federated identifier was just established.

718 • SHOULD, if the <AddEntityRequest> message contained a <CreatePSObject> element, attempt to create
719 or verify the existence of an object for the inviting user in the PS of the invited user (when made possible by a
720 federated identifier being established for the invited user).

721 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
722 sequence initiated "from the other side." The PS of the inviting user MUST ensure that no duplicate object be
723 added.

724 MAY, in order to determine whether an object for the inviting user already exists, query the members of the PS
725 of the invited user using the <ListMembersRequest> message and ask the invited user to assist in determining
726 whether the inviting user is already in the list. Other mechanisms (e.g., using a <TestMembershipRequest>)
727 for making this determination MAY alternatively be used.

728 SHOULD, if there is no existing object for the inviting user, request that an object be created with either the
729 <AddEntityRequest> or <AddKnownEntityRequest> messages.

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

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

736 This <ims:IdentityMappingRequest> message to the IDP MUST include any policy directives present in the
737 <AddEntityRequest>.

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

742 3.10. Adding a Known Entity

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

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

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

756 3.10.1. wsa:Action Values

757 <AddKnownEntityRequest> messages MUST include a <wsa:Action> SOAP header with the value of
758 "urn:liberty:ps:2006-08:AddKnownEntityRequest." <AddKnownEntityResponse> messages MUST include a
759 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:AddKnownEntityResponse."

760 3.10.2. AddKnownEntityRequest Message

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

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

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

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

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

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

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

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

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

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

```
781
782 <!-- Declaration of AddKnownEntityRequest element -->
783 <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType"/>
784 <!-- Definition of AddKnownEntityRequestType -->
785 <xs:complexType name="AddKnownEntityRequestType">
786   <xs:complexContent>
787     <xs:extension base="RequestAbstractType">
788       <xs:sequence>
789         <xs:element ref="Object"/>
790         <xs:element ref="sec:Token"/>
791         <xs:element ref="CreatePSObject" minOccurs="0"/>
792         <xs:element ref="Subscription" minOccurs="0"/>
793         <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
794       </xs:sequence>
795     </xs:extension>
796   </xs:complexContent>
797 </xs:complexType>
798
```

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

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

```
811     </sec:Token>
812 </AddKnownEntityRequest>
813
```

814 3.10.3. AddKnownEntityResponse Message

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

817 The `<AddKnownEntityResponse>` message has the complex type **AddKnownEntityType**, which ex-
818 tends **ResponseAbstractType** and adds the following elements:

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

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

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

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

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

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

```
837
838 <!-- Declaration of AddKnownEntityResponse element -->
839 <xs:element name="AddKnownEntityResponse" type="AddKnownEntityType"/>
840 <!-- Definition of AddKnownEntityType -->
841 <xs:complexType name="AddKnownEntityType">
842   <xs:complexContent>
843     <xs:extension base="ResponseAbstractType">
844       <xs:sequence>
845         <xs:element ref="Object" minOccurs="0"/>
846         <xs:element ref="SptoPSRedirectURL" minOccurs="0" maxOccurs="1"/>
847         <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
848       </xs:sequence>
849     </xs:extension>
850   </xs:complexContent>
851 </xs:complexType>
852
```

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

```
856 <AddKnownEntityResponse>
857   <Status code="OK"/>
858   <Object NodeType="urn:liberty:ps:entity">
```

```
860         <ObjectID>https://ps.com/lafnervf</ObjectID>
861         <DisplayName>Bob</DisplayName>
862     </Object>
863 </AddKnownEntityResponse>
864
```

865 3.10.4. Processing Rules

866 The WSC:

- 867 • MUST include an <Object> element within the <AddKnownEntityRequest> message.
- 868 • MUST include a `NodeType` attribute on the <Object> element with a value of `urn:liberty:ps:entity`.
- 869 • MUST include a <Token> element within the <AddKnownEntityRequest> message.
870 When the token is not an identity token (as is the likely case when the known identifier is provided by the inviting
871 user), the WSC SHOULD use a SAML <saml:NameID> element within the <Token> element. If the WSC
872 knows the format of the known identifier, it SHOULD use the appropriate value for the `Format` attribute on the
873 <saml:NameID> element.
- 874 • MUST, if a SAML <Assertion> is used as the identity token format, specify the known identifier in that
875 assertion's <Subject> element.
- 876 • MUST include at least one <DisplayName> for the invited user within the <Object> element. This element
877 contains the friendly name that the user desires be used for the created user Object.
- 878 • MAY, if it desires that the PS provider create (or verify the existence of) an Object for the inviting user at the PS
879 provider of the invited user, include a <CreatePSObject> element.
- 880 • MAY, if it desires that an identity token be returned to it through a subsequent <Notification> message, include
881 a <Subscription> element. The presence of a <Subscription> element indicates to the PS provider that the
882 SP desires that the PS provider return to it (when later possible) an identity token for the invited user within a
883 <Notification> message - this possible after a federation has been established between the PS provider and the
884 appropriate IDP. If no <Subscription> element is present, the SP is indicating that the PS provider need not
885 return an identity token through this mechanism.

886 In responding to an <AddKnownEntityRequest> message, the PS provider:

- 887 • MAY include either or both of a <SPtoPSRedirectURL> or <QueryString> element in the
888 <AddKnownEntityResponse> message returned to the calling SP.
- 889 • SHOULD, if the <AddKnownEntityRequest> message contained a <Subscription> element, send a
890 <ims:IdentityMappingRequest> message to that IDP requesting an identity token for the user for which the
891 federated identifier was established but in the namespace of the requesting SP.
892 This <ims:IdentityMappingRequest> message to the IDP MUST include any policy directives present in the
893 <AddKnownEntityRequest>.
- 894 • SHOULD, if the <AddKnownEntityRequest> message contained a <Subscription> element and the
895 <ims:IdentityMappingRequest> message to the IDP resulted in an identity token for the user being returned,
896 forward on this identity token to the SP within a <Notification> message corresponding to the original
897 <Subscription> element.

898 • SHOULD, if the <AddKnownEntityRequest> message contained a <CreatePSObject> element, ensure that
899 there be an object for the inviting user in the PS of the invited user.

900 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
901 initiated "from the other side." The PS of the inviting user MUST ensure that no duplicate object be added.

902 SHOULD, in order to determine whether an object for the inviting user already exists, query the members of the
903 PS of the invited user using the <ListMembersRequest> message.

904 SHOULD, if there is no existing object for the inviting user, request that an object be created with either the
905 <AddEntityRequest> or <AddKnownEntityRequest> messages.

906 SHOULD, if sending a <AddKnownEntityRequest> message for the addition, include a <sec:Token> element
907 carrying a token for the inviting user.

908 3.11. Removing an Entity

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

912 3.11.1. wsa:Action Values

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

916 3.11.2. RemoveEntityRequest Message

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

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

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

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

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

```
928 <!-- Declaration of RemoveEntityRequest element -->
929 <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType" />
930 <!-- Definition of RemoveEntityRequestType -->
931 <xs:complexType name="RemoveEntityRequestType">
932   <xs:complexContent>
933     <xs:extension base="RequestAbstractType">
934       <xs:sequence>
935         <xs:element ref="TargetObjectID" maxOccurs="unbounded" />
936       </xs:sequence>
937     </xs:extension>
938   </xs:complexContent>
939 </xs:complexType>
940
941
```

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

```
943
944 <RemoveEntityRequest>
945   <TargetObjectID>https://ps.com/lafnervf</TargetObjectID>
946 </RemoveEntityRequest>
947
```

948 3.11.3. RemoveEntityResponse Message

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

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

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

```
954 <!-- Declaration of RemoveEntityResponse element -->
955 <xs:element name="RemoveEntityResponse" type="ResponseAbstractType" />
956
957
```

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

```
960
961 <RemoveEntityResponse>
962   <Status code="OK" />
963 </RemoveEntityResponse>
964
```

965 3.11.4. Processing Rules

966 The WSC:

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

968 The PS provider:

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

972 • NotDirectChild

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

974 3.12. Adding a Collection

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

977 3.12.1. wsa:Action Values

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

981 3.12.2. AddCollectionRequest Message

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

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

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

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

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

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

```
992
993 <!-- Declaration of AddCollectionRequest element -->
994 <xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
995 <!-- Definition of AddCollectionRequestType -->
996 <xs:complexType name="AddCollectionRequestType">
997   <xs:complexContent>
998     <xs:extension base="RequestAbstractType">
999       <xs:sequence>
1000         <xs:element ref="Object"/>
1001         <xs:element ref="Subscription" minOccurs="0"/>
1002       </xs:sequence>
1003     </xs:extension>
1004   </xs:complexContent>
1005 </xs:complexType>
1006
```

1007 The following is an example of an <AddCollectionRequest> message used to create an Object for a group.

```
1008
1009 <AddCollectionRequest>
1010   <Object NodeType="urn:liberty:ps:collection">
1011     <DisplayName>Soccer Team</DisplayName>
1012   </Object>
1013 </AddCollectionRequest>
1014
```

1015 3.12.3. AddCollectionResponse Message

1016 A PS provider responds to an `<AddCollectionRequest>` message with an `<AddCollectionResponse>` message
1017 containing the newly created `<Object>` element.

1018 The `<AddCollectionResponse>` message has the complex type `AddCollectionResponseType`, which extends
1019 `ResponseAbstractType` and adds the following element:

1020 `<Object>` [**Optional**] The `<Object>` element is used to convey the `Object` element just created at the PS provider.

1021 The schema declaration for the `<AddCollectionResponse>` message is shown below.

```
1022  
1023 <!-- Declaration of AddCollectionResponse element -->  
1024 <xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>  
1025 <!-- Definition of AddCollectionResponseType -->  
1026 <xs:complexType name="AddCollectionResponseType">  
1027   <xs:complexContent>  
1028     <xs:extension base="ResponseAbstractType">  
1029       <xs:sequence>  
1030         <xs:element ref="Object" minOccurs="0"/>  
1031       </xs:sequence>  
1032     </xs:extension>  
1033   </xs:complexContent>  
1034 </xs:complexType>  
1035
```

1036 The following is an example of an `<AddCollectionResponse>` to the `<AddCollectionRequest>` message above.
1037 The PS provider is responding that the request that the Soccer Team be added was successful and returns the created
1038 `<Object>` element.

```
1039  
1040 <AddCollectionResponse>  
1041   <Status code="OK"/>  
1042   <Object NodeType="urn:liberty:ps:collection">  
1043     <ObjectID>https://ps.com/roqlsfof</ObjectID>  
1044     <DisplayName>Soccer Team</DisplayName>  
1045   </Object>  
1046 </AddCollectionResponse>  
1047
```

1048 3.12.4. Processing Rules

1049 The WSC:

- 1050 • **MUST** include an `<Object>` element within the `<AddCollectionRequest>` message.
- 1051 • **MUST** include a `NodeType` attribute on the `<Object>` element with a value of `urn:liberty:ps:collection`.
- 1052 • **MUST** include at least one `<DisplayName>` element for a group within the `<Object>`. This element contains the
1053 friendly name that the user desires be used for the created group `Object`.

1054 In responding to an `<AddCollectionRequest>` message, the PS provider:

- 1055 • **MUST** return an `<Object>` element within the `<AddCollectionResponse>` message with the same
1056 `<DisplayName>` as specified on the `<AddCollectionRequest>`.
- 1057 • **MUST** include an `<ObjectID>` element for the newly created group `Object`.

3.13. Removing a Collection

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

3.13.1. wsa:Action Values

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

3.13.2. RemoveCollectionRequest Message

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

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

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

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

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

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

```
<!-- Declaration of RemoveCollectionRequest element -->
<xs:element name="RemoveCollectionRequest" type="RemoveCollectionRequestType"/>
<!-- Definition of RemoveCollectionRequestType -->
<xs:complexType name="RemoveCollectionRequestType" >
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

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

```
<RemoveCollectionRequest>
  <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
</RemoveCollectionRequest>
```

3.13.3. RemoveCollectionResponse Message

1102 A PS provider responds to a <RemoveCollectionRequest> message with a <RemoveCollectionResponse>
1103 element. A PS provider removes the specified <Object> and responds a status of this process based on the processing
1104 rules described in [Section 3.11.4](#).

1105 The <RemoveCollectionResponse> message has the type of **ResponseAbstractType**

1106 The schema declaration for the <RemoveCollectionResponse> message is shown below.

```
1107 <!-- Declaration of RemoveCollectionResponse element -->  
1108 <xs:element name="RemoveCollectionResponse" type="ResponseAbstractType"/>  
1109  
1110
```

1111 The following is an example of a <RemoveCollectionResponse> to the <RemoveCollectionRequest> message
1112 above. The PS provider is responding that the request that a specified Object be removed was successful.

```
1113  
1114 <RemoveCollectionResponse>  
1115   <Status code="OK" />  
1116 </RemoveCollectionResponse>  
1117
```

1118 **3.13.4. Processing Rules**

1119 The WSC:

- 1120 • MUST include a `NodeType` attribute on the <Object> element with a value of *urn:liberty:ps:collection*.

1121 The PS provider:

- 1122 • MUST remove the specified <Object> from the PS list.
- 1123 • MUST NOT, if the specified group <Object> has one or more child <Object> elements, remove any the child
1124 Objects unless those child <Object>s are explicitly specified by their own <ObjectID> values in separate
1125 <TargetObjectID> elements.

1126 **3.14. Adding to a Collection**

1127 A WSC uses the <AddToCollectionRequest> message to request that child Object elements be added to
1128 an existing group Object. Both user and group Objects can be added to a parent group Object with the
1129 <AddToCollectionRequest> message.

1130 **3.14.1. wsa:Action Values**

1131 <AddToCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1132 "urn:liberty:ps:2006-08:AddToCollectionRequest." <AddToCollectionResponse> messages MUST include a
1133 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:AddToCollectionResponse."

1134 **3.14.2. AddToCollectionRequest Message**

1135 The target parent group Object to which child Objects are to be added is indicated by the value of the
1136 <TargetObjectID> element within the <AddToCollectionRequest> element. The child Objects being added
1137 are specified by the values of the <ObjectID> elements within the <AddToCollectionRequest> element.

1138 The <AddToCollectionRequest> message has the complex type **AddToCollectionRequestType**, which ex-
1139 tends **RequestAbstractType** and adds the following elements:

1140 <TargetObjectID> **[Required]** The <TargetObjectID> element is used to convey the ObjectID of the target
1141 group Object to which specified Objects are added.

1142 <ObjectID> **[Required]** The <ObjectID> element is used to convey ObjectIDs of the Objects to be added to
1143 the target group Object.

1144 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
1145 desires that the PS provider send a notification if and when the membership of the targeted
1146 group changes.

1147 The schema declaration for the <AddToCollectionRequest> message is shown below.

```
1148  
1149 <!-- Declaration of AddToCollectionRequest element -->  
1150 <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>  
1151 <!-- Definition of AddToCollectionRequestType -->  
1152 <xs:complexType name="AddToCollectionRequestType">  
1153   <xs:complexContent>  
1154     <xs:extension base="RequestAbstractType">  
1155       <xs:sequence>  
1156         <xs:element ref="TargetObjectID"/>  
1157         <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>  
1158         <xs:element ref="Subscription" minOccurs="0"/>  
1159       </xs:sequence>  
1160     </xs:extension>  
1161   </xs:complexContent>  
1162 </xs:complexType>  
1163
```

1164 The following is an example of an <AddToCollectionRequest> message used to add three Objects to the target
1165 group Object.

```
1166  
1167 <AddToCollectionRequest>  
1168   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>  
1169   <ObjectID>https://ps.com/qaf3eflo</ObjectID>  
1170   <ObjectID>https://ps.com/bzpfnrns</ObjectID>  
1171   <ObjectID>https://ps.com/nsdflfss</ObjectID>  
1172 </AddToCollectionRequest>  
1173
```

1174 3.14.3. AddToCollectionResponse Message

1175 A PS provider responds to an <AddToCollectionRequest> message with an <AddToCollectionResponse>
1176 message. The PS provider makes the indicated modification to the specified target <Object> and responds with the
1177 status.

1178 The <AddToCollectionResponse> message has the type of **ResponseAbstractType**

1179 The schema declaration for the <AddToCollectionResponse> message is shown below.

```
1180  
1181 <!-- Declaration of AddToCollectionResponse element -->  
1182 <xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>  
1183
```

1184 The following is an example of an <AddToCollectionResponse> to the <AddToCollectionRequest> message
1185 above. The PS provider is responding that the request that the three Objects be added to the target Object was
1186 successful.

```
1187  
1188 <AddToCollectionResponse>
```

```
1189     <Status code="OK" />
1190 </AddToCollectionResponse>
1191
```

1192 3.14.4. Processing Rules

1193 The WSC:

- 1194 • MUST specify, as a value of the <TargetObjectID>, an ObjectID of the Object that has
1195 *urn:liberty:ps:collection* as a value of Node**Type** attribute.

1196 The PS provider:

- 1197 • MUST, if the Object specified by the value of the <TargetObjectID> element has *urn:liberty:ps:entity* as the
1198 value of its Node**Type** attribute, respond with *Failed* as the code attribute of the top level <lu:Status> element.
1199 A second level <lu:Status> MAY be inserted. If so, the code attribute of that second level <lu:Status>
1200 element MUST be set with the following status code:

- 1201 • ObjectIsEntity

- 1202 • MUST, if the Object specified by the value of an <ObjectID> element is already a member of the targeted
1203 collection, respond with *Failed* as the code attribute of the top level <lu:Status> element. A second level
1204 <lu:Status> MAY be inserted. If so, the code attribute of that second level <lu:Status> element MUST be
1205 set with the following status code:

- 1206 • DuplicateObject

- 1207 • MUST NOT allow the creation of circular collections. A circular collection is one which includes, at any layer
1208 of the structure below it, a reference to the same collection such that a dereference of the collection would result
1209 in an infinite loop. If the PS provider receives an <AddToCollectionRequest> message that would result in a
1210 circular collection, it MUST respond with *Failed* as the code attribute of the top level <lu:Status> element. A
1211 second level <lu:Status> MAY be inserted. If so, the code attribute of that second level <lu:Status> element
1212 MUST be set with the following status code:

- 1213 • CircularCollection

1214 3.15. Removing from a Collection

1215 A WSC uses the <RemoveFromCollectionRequest> message to request the removal of a child Object element
1216 from a parent group Object. Both user and group Objects can be removed from a parent group Object with the
1217 <RemoveFromCollectionRequest> message.

1218 3.15.1. wsa:Action Values

1219 <RemoveFromCollectionRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1220 "urn:liberty:ps:2006-08:RemoveFromCollectionRequest." <RemoveFromCollectionResponse> messages MUST
1221 include a <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:RemoveFromCollectionResponse."

1222 3.15.2. RemoveFromCollectionRequest Message

1223 The target parent group Object from which a child Object is to be removed is indicated by the value of the
1224 <TargetObjectID> element within the <RemoveFromCollectionRequest> message. The child Object being
1225 removed are specified by the value(s) of the <ObjectID> elements within the <RemoveFromCollectionRequest>
1226 element.

1227 The <RemoveFromCollectionRequest> message has the complex type **RemoveFromCollectionRequestType**,
1228 which extends **RequestAbstractType** and adds the following elements:

1229 <TargetObjectID> **[Required]** The <TargetObjectID> element is used to convey the ObjectID of the target
 1230 group Object from which specified Objects are to be removed.

1231 <ObjectID> **[Required]** The <ObjectID> element is used to convey ObjectIDs of the Objects that would be
 1232 removed from the target group Object.

1233 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the WSC
 1234 desires that the PS provider send a notification if and when the membership of the targeted
 1235 group changes.

1236 The schema declaration for the <RemoveFromCollectionRequest> message is shown below.

```
1237
1238 <!-- Declaration of RemoveFromCollectionRequest element -->
1239 <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
1240 <!-- Definition of RemoveFromCollectionRequestType -->
1241 <xs:complexType name="RemoveFromCollectionRequestType">
1242   <xs:complexContent>
1243     <xs:extension base="RequestAbstractType">
1244       <xs:sequence>
1245         <xs:element ref="TargetObjectID"/>
1246         <xs:element ref="ObjectID" maxOccurs="unbounded"/>
1247         <xs:element ref="Subscription" minOccurs="0"/>
1248       </xs:sequence>
1249     </xs:extension>
1250   </xs:complexContent>
1251 </xs:complexType>
1252
```

1253 The following is an example of a <RemoveFromCollectionRequest> message used to remove three Objects from
 1254 the target group Object.

```
1255
1256 <RemoveFromCollectionRequest>
1257   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
1258   <ObjectID>https://ps.com/qaf3eflo</ObjectID>
1259   <ObjectID>https://ps.com/bzpfnrns</ObjectID>
1260   <ObjectID>https://ps.com/nsdfllfss</ObjectID>
1261 </RemoveFromCollectionRequest>
1262
```

1263 3.15.3. RemoveFromCollectionResponse Message

1264 A PS provider responds to a <RemoveFromCollectionRequest> message with a <RemoveFromCollectionResponse>
 1265 message. The PS provider makes the indicated modification to the specified target Object and responds with the
 1266 status.

1267 The <RemoveFromCollectionResponse> message has the type of **ResponseAbstractType**

1268 The schema declaration for the <RemoveFromCollectionResponse> message is shown below.

```
1269
1270 <!-- Declaration of RemoveFromCollectionResponse element -->
1271 <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>
1272
```

1273 The following is an example of a <RemoveFromCollectionResponse> to the <RemoveFromCollectionRequest>
 1274 message above. The PS provider is responding that the request that the three objects be removed from the target
 1275 Object was successful.

```
1276
1277 <RemoveFromCollectionResponse>
```

```
1278     <Status code="OK" />
1279 </RemoveFromCollectionResponse>
1280
```

1281 3.15.4. Processing Rules

1282 The WSC:

- 1283 • MUST specify, as a value of the <TargetObjectID>, an ObjectID of the Object that has
1284 *urn:liberty:ps:collection* as a value of Node**Type** attribute.

1285 The PS provider:

- 1286 • MUST, if the Object specified by the value of the <TargetObjectID> element has *urn:liberty:ps:entity* as the
1287 value of its Node**Type** attribute, respond with *Failed* as the code attribute of the top level <lu:Status> element.
1288 A second level <lu:Status> MAY be inserted. If so, the code attribute of that second level <lu:Status>
1289 element MUST be set with the following status code:

- 1290 • ObjectIsEntity

- 1291 • MUST, if the Object specified by the value of an <ObjectID> element is not a member of the targeted collection,
1292 respond with *Failed* as the code attribute of the top level <lu:Status> element. A second level <lu:Status>
1293 MAY be inserted. If so, the code attribute of that second level <lu:Status> element MUST be set with the
1294 following status code:

- 1295 • CannotFindObject

1296 3.16. Listing Members

1297 A WSC uses the <ListMembersRequest> message to navigate the Object structure of the users
1298 (*urn:liberty:ps:entity* Objects) and groups (*urn:liberty:ps:collection* Objects) that comprise the PS resource.

1299 A WSC can control how Objects are returned by specifying its preferences with the Structured attribute.

1300 If a WSC does not specify a <TargetObjectID> element in the <ListMembersRequest> message, the default
1301 targeted Object is the root node.

1302 3.16.1. wsa:Action Values

1303 <ListMembersRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1304 "urn:liberty:ps:2006-08:ListMembersRequest." <ListMembersResponse> messages MUST include a
1305 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:ListMembersResponse."

1306 3.16.2. ListMembersRequest Message

1307 The WSC indicates to the PS provider the Object of interest by specifying that <Object> element's ObjectID in
1308 the <TargetObjectID> element. If no <TargetObjectID> element is specified in the <ListMembersRequest>
1309 message, the PS provider MUST return all the top-level Objects (i.e., Objects that do not have any parent Object.)

1310 The <ListMembersRequest> message has the complex type **ListMembersRequestType**, which extends
1311 **RequestAbstractType** and adds the following attributes and elements:

1312 Structured [**Optional**] The Structured allows a WSC to indicate what portion of the Object tree structure it
1313 desires be returned. See [Section 3.16.2.1](#) for more detail.

1314 **Count [Optional]** The `Count` attribute specifies how many child `<Object>` elements should be returned in a
1315 `<ListMembersResponse>` message. See [Section 3.16.2.2](#) for more detail.

1316 **Offset [Optional]** The `Offset` attribute specifies from which element to continue, when requesting more data.
1317 See [Section 3.16.2.2](#) for more detail.

1318 `<TargetObjectID>` **[Optional]** The `<TargetObjectID>` element is used to convey an `ObjectID` of the target
1319 group `Object` whose child `Objects` are to be listed.

1320 `<Subscription>` **[Optional]** The `<Subscription>` element is used to indicate to the PS provider that the WSC
1321 desires that the PS provider send a notification if and when the membership of the targeted
1322 group changes.

1323 Interpretation of the membership for which a change `<Notify>` message should be sent will
1324 depend on the value of the `Structured` attribute on the `<ListMembersRequest>` message.
1325 For instance, if the attribute has a value of "tree," the subscription corresponds to the full
1326 object tree and `<Notify>` messages **MUST** be sent accordingly.

1327 For a subscription within a `<ListMembersRequest>` message, the PS provider **MUST**
1328 assess "changes" against whatever was returned in the original `<ListMembersResponse>`.
1329 A `<Notify>` **MUST** be sent if, were the WSC to resend the same request, the results would
1330 be different than originally sent.

1331 The schema declaration for the `<ListMembersRequest>` message is shown below.

```
1332
1333 <!-- Declaration of ListMembersRequest element -->
1334 <xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
1335 <!-- Definition of ListMembersRequestType -->
1336 <xs:complexType name="ListMembersRequestType">
1337   <xs:complexContent>
1338     <xs:extension base="RequestAbstractType">
1339       <xs:sequence>
1340         <xs:element ref="TargetObjectID" minOccurs="0"/>
1341         <xs:element ref="Subscription" minOccurs="0"/>
1342       </xs:sequence>
1343       <xs:attribute name="Structured" type="xs:string" use="optional"/>
1344       <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
1345       <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
1346     </xs:extension>
1347   </xs:complexContent>
1348 </xs:complexType>
1349
```

1350 3.16.2.1. Structured Attribute

1351 WSCs may choose to navigate the hierarchal tree structure for a given `Object` incrementally (i.e., by successive
1352 requests to probe deeper) or to have the complete tree returned. The `Structured` attribute allows the WSC to indicate
1353 this preference to the PS provider. Additionally, the WSC uses the `Structured` attribute to indicate whether or not it
1354 desires to see collection `Object` elements returned or only entity `Object` elements.

1355 The `Structured` attribute takes three possible values

1356 *children* The WSC is indicating that it desires only the direct child collection and entity objects of the
1357 targeted object.

1358 The default value for the `Structured` attribute is *children*.

1359 *tree* the WSC is indicating that it desires the full tree structure of the targetted object.

1360 *entities* the WSC is indicating that it desires a flat view of the full tree structure, e.g., one in which any
1361 collection hierarchy is hidden.

1362 3.16.2.2. Count and Offset Attributes

1363 When the specified Object has multiple child Objects, the WSC may desire to be sent the child <Object> elements
1364 in smaller sets (i.e., a smaller number of elements at a time). This is achieved by using the Count and Offset attributes
1365 of the <ListMembersRequest> element.

1366 The Count attribute defines how many child <Object> elements should be returned in a <ListMembersResponse>
1367 message. The Count attribute only pertains to the direct child <Object> elements of the Object specified in the
1368 <ListMembersRequest> message.

1369 The Offset attribute specifies from which element to continue, when requesting more data. The default value is zero,
1370 which refers to the first child <Object> element.

1371 3.16.3. ListMembersResponse Message

1372 A PS provider responds to a <ListMembersRequest> message with a <ListMembersResponse> message contain-
1373 ing the appropriate set of <Object> elements.

1374 The <ListMembersResponse> message has the complex type **ListMembersResponseType**, which extends
1375 **ResponseAbstractType** and adds the following element:

1376 <Object> [**Optional**] The <Object> element is used to convey data of zero or more Objects to be listed.

1377 The schema declaration for the <ListMembersResponse> message is shown below.

```
1378
1379 <!-- Declaration of ListMembersResponse element -->
1380 <xs:element name="ListMembersResponse" type="ListMembersResponseType" />
1381 <!-- Definition of ListMembersResponseType -->
1382 <xs:complexType name="ListMembersResponseType">
1383   <xs:complexContent>
1384     <xs:extension base="ResponseAbstractType">
1385       <xs:sequence>
1386         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded" />
1387       </xs:sequence>
1388     </xs:extension>
1389   </xs:complexContent>
1390 </xs:complexType>
1391
```

1392 3.16.4. Examples

1393 All the examples described in this subsection assume that a PS provider has the following virtual XML document for
1394 some user's PS list.

```
1395
1396 <Object NodeType="urn:liberty:ps:entity">
1397   <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1398   <DisplayName>Mary</DisplayName>
1399 </Object>
1400 <Object NodeType="urn:liberty:ps:entity">
1401   <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1402   <DisplayName>Bob</DisplayName>
1403 </Object>
1404 <Object NodeType="urn:liberty:ps:entity">
1405   <ObjectID>https://ps.com/reremvls</ObjectID>
1406   <DisplayName>Nick</DisplayName>
1407 </Object>
```

```

1408 <Object NodeType="urn:liberty:ps:entity">
1409   <ObjectID>https://ps.com/soijfsfd</ObjectID>
1410   <DisplayName>JoJo</DisplayName>
1411 </Object>
1412 <Object NodeType="urn:liberty:ps:entity">
1413   <ObjectID>https://ps.com/sdosafms</ObjectID>
1414   <DisplayName>Taro</DisplayName>
1415 </Object>
1416 <Object NodeType="urn:liberty:ps:entity">
1417   <ObjectID>https://ps.com/lgsdfsf</ObjectID>
1418   <DisplayName>Hanako</DisplayName>
1419 </Object>
1420 <Object NodeType="urn:liberty:ps:collection">
1421   <ObjectID>https://ps.com/eiruvoie</ObjectID>
1422   <DisplayName>Soccer Team</DisplayName>
1423   <Object NodeType="urn:liberty:ps:collection">
1424     <ObjectID>https://ps.com/nmerflas</ObjectID>
1425     <DisplayName>Starting Members</DisplayName>
1426     <ObjectRef Ref="https://ps.com/lsdjfojd"/>
1427     <ObjectRef Ref="https://ps.com/sijfsfsf"/>
1428   </Object>
1429   <ObjectRef Ref="https://ps.com/reremvls"/>
1430   <ObjectRef Ref="https://ps.com/soijfsfd"/>
1431 </Object>
1432 <Object NodeType="urn:liberty:ps:collection">
1433   <ObjectID>https://ps.com/zxlifdf</ObjectID>
1434   <DisplayName>Family</DisplayName>
1435   <ObjectRef Ref="https://ps.com/sdosafms"/>
1436   <ObjectRef Ref="https://ps.com/lgsdfsf"/>
1437 </Object>
1438

```

1439 The following is an example of a <ListMembersRequest> message by which a WSC requests the list of members
1440 of the "Soccer Team" collection Object. As the WSC specifies Structured="entities", it is indicating that it
1441 desires to have a "flat" view of that group's Object tree returned, e.g., one in which all collection hierarchy is removed.

```

1442
1443 <ListMembersRequest Structured="entities">
1444   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1445 </ListMembersRequest>
1446

```

1447 The following is an example <ListMembersResponse> message as might be returned to the
1448 <ListMembersRequest> above.

```

1449
1450 <ListMembersResponse>
1451   <Status code="OK"/>
1452   <Object NodeType="urn:liberty:ps:entity">
1453     <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1454     <DisplayName>Mary</DisplayName>
1455   </Object>
1456   <Object NodeType="urn:liberty:ps:entity">
1457     <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1458     <DisplayName>Bob</DisplayName>
1459   </Object>
1460   <Object NodeType="urn:liberty:ps:entity">
1461     <ObjectID>https://ps.com/reremvls</ObjectID>
1462     <DisplayName>Nick</DisplayName>
1463   </Object>
1464   <Object NodeType="urn:liberty:ps:entity">
1465     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1466     <DisplayName>JoJo</DisplayName>
1467   </Object>
1468 </ListMembersResponse>
1469

```

1470 As the WSC indicated it desired a flat view, the PS expanded the group Object called "Starting Members" and returns
 1471 its two child entity Object elements (for Mary and Bob) instead of the collection Object itself.

1472 Alternatively, the following is a <ListMembersResponse> message as might be returned to a
 1473 <ListMembersRequest> message in which the WSC indicated it desired to see the full tree structure by
 1474 specifying Structured="tree".

```

1475 <ListMembersResponse>
1476   <Status code="OK" />
1477   <Object NodeType="urn:liberty:ps:collection">
1478     <ObjectID>https://ps.com/nmerflas</ObjectID>
1479     <DisplayName>Starting Members</DisplayName>
1480     <Object NodeType="urn:liberty:ps:entity">
1481       <ObjectID>https://ps.com/lsdjfojd</ObjectID>
1482       <DisplayName>Mary</DisplayName>
1483     </Object>
1484     <Object NodeType="urn:liberty:ps:entity">
1485       <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1486       <DisplayName>Bob</DisplayName>
1487     </Object>
1488   </Object>
1489   <Object NodeType="urn:liberty:ps:entity">
1490     <ObjectID>https://ps.com/reremvls</ObjectID>
1491     <DisplayName>Nick</DisplayName>
1492   </Object>
1493   <Object NodeType="urn:liberty:ps:entity">
1494     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1495     <DisplayName>JoJo</DisplayName>
1496   </Object>
1497 </ListMembersResponse>
1498
1499
```

1500 The PS returns the full sub-tree for the specified target object including the hierarchy of the "Starting Members" group.

1501 Lastly, the following is an example <ListMembersResponse> message as might be returned to a
 1502 <ListMembersRequest> message in which the WSC indicated it desired to see only direct children by spec-
 1503 ifying Structured="children".

```

1504 <ListMembersResponse>
1505   <Status code="OK" />
1506   <Object NodeType="urn:liberty:ps:collection">
1507     <ObjectID>https://ps.com/nmerflas</ObjectID>
1508     <DisplayName>Starting Members</DisplayName>
1509   </Object>
1510   <Object NodeType="urn:liberty:ps:entity">
1511     <ObjectID>https://ps.com/reremvls</ObjectID>
1512     <DisplayName>Nick</DisplayName>
1513   </Object>
1514   <Object NodeType="urn:liberty:ps:entity">
1515     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1516     <DisplayName>JoJo</DisplayName>
1517   </Object>
1518 </ListMembersResponse>
1519
1520
```

1521 3.16.5. Processing Rules

- 1522 • The WSC SHOULD NOT include a <Subscription> element in a <ListMembersRequest> message if the
 1523 Offset attribute has any value other than "0."

- 1524 • The PS provider MUST, if the `<ListMembersRequest>` contains a Subscription and the Offset attribute has any
1525 value other than "0," and it is otherwise capable of returning results, return those results as indicated by the Count
1526 and Offset attributes, but still reject the Subscription by responding with "OKButNoSubscription" as the code
1527 attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code
1528 attribute of that second level `<lu:Status>` element MUST be set with the following status code:
- 1529 • NoSubscribeWithOffset
- 1530 • The PS provider SHOULD dereference all `<ObjectRef>` elements and replace them with the appropriate
1531 `<Object>` elements before returning.
- 1532 • At any one level of the tree, the PS provider SHOULD remove all duplicate `<Object>` elements before returning.
- 1533 • If the Structured attribute is set as *tree*, the PS provider MUST return all the direct child and descendant
1534 `<Object>` elements of the specified Object.
- 1535 • If the Structured attribute is not set in the request, the PS provider MUST process it as if it is set as *children*.
- 1536 • If the Structured attribute is set as *entities*, the PS provider MUST return all the direct child and descendant
1537 entity `<Object>` elements of the specified Object (subject to the restriction defined by the Count attribute, if
1538 present). Any collection `<Object>` elements MUST be removed and only entity `<Object>` elements returned.
- 1539 • If the Structured attribute is set as *children*, the PS provider MUST return all the direct child collection and
1540 entity `<Object>` elements of the specified Object.
- 1541 • If the Count attribute is included in a `<ListMembersRequest>` message, the PS provider SHOULD NOT respond
1542 with more objects than specified. A PS provider MAY return a smaller number of objects than specified by the
1543 Count attribute.
- 1544 • If the Offset attribute is included in a `<ListMembersRequest>` message, the PS provider SHOULD respond with
1545 a list of `<Object>` elements, the first in the list being that `<Object>` element whose position in the complete list
1546 is specified by the value of the Offset attribute.
- 1547 • If a PS provider receives a `<ListMembersRequest>` message on which the value of `<TargetObjectID>`
1548 matches that of an `<ObjectID>` element of a given Object, and if the value of the `NodeType` attribute of the
1549 matched Object is *urn:liberty:ps:entity*, then the PS provider MUST respond with *Failed* as the code attribute of
1550 the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of
1551 that second level `<lu:Status>` element MUST be set with the following status code:
- 1552 • ObjectIsEntity
- 1553 • The PS Provider MUST, if unable to find the targeted Object, respond with *Failed* as the code attribute of the
1554 top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that
1555 second level `<lu:Status>` element MUST be set with the following status code:
- 1556 • CannotFindObject
- 1557 • When the targeted Object has no child Objects, the PS provider MUST respond with *OK* as the code attribute of
1558 the top level `<lu:Status>` element with no `<Object>` element in the response.
- 1559 • The PS provider SHOULD, if the `<ListMembersRequest>` contains a Subscription and the Structured attribute
1560 has any value other than "children," and it is otherwise capable of returning results, return those results but
1561 still reject the Subscription by responding with "OKButNoSubscription" as the code attribute of the top level
1562 `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second
1563 level `<lu:Status>` element MUST be set with the following status code:
- 1564 • SubscribeToChildrenOnly

1565 An "OKButNoSubscription" Status code SHOULD NOT be interpreted as reflecting a PS provider's overall ability
1566 to support subscriptions, rather simply its unwillingness to accept the particular subscription requested.

1567 3.17. Retrieving Info

1568 A WSC uses the <GetObjectInfoRequest> message to retrieve information for a particular Object. An Object's
1569 information includes all the child elements and attributes of the <Object> element, except for either <Object> or
1570 <ObjectRef> elements.

1571 Note that if a WSC wants to get a child members' Objects of the particular Object, the WSC MUST use
1572 <ListMembersRequest> message (see [Section 3.16](#)).

1573 3.17.1. wsa:Action Values

1574 <GetObjectInfoRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1575 "urn:liberty:ps:2006-08:GetObjectInfoRequest." <GetObjectInfoResponse> messages MUST include a
1576 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:GetObjectInfoResponse."

1577 3.17.2. GetObjectInfoRequest Message

1578 The WSC indicates to the PS provider the Object of interest by specifying that <Object> element's ObjectID in
1579 the <TargetObjectID> element.

1580 The <GetObjectInfoRequest> message has the complex type **GetObjectInfoRequestType**, which extends
1581 **RequestAbstractType** and adds the following element:

1582 <TargetObjectID> **[Required]** The <TargetObjectID> element is used to convey an ObjectID of the target
1583 Object of which information is requested.

1584 <Subscription> **[Optional]** The <Subscription> element is used to indicate to the PS provider that the
1585 WSC desires that the PS provider send a notification if and when the information (but not
1586 membership) of the targeted Object changes.

1587 The schema declaration for the <GetObjectInfoRequest> message is shown below.

```
1588 <!-- Declaration of GetObjectInfoRequest element -->  
1589 <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType" />  
1590 <!-- Definition of GetObjectInfoRequestType -->  
1591 <xs:complexType name="GetObjectInfoRequestType">  
1592   <xs:complexContent>  
1593     <xs:extension base="RequestAbstractType">  
1594       <xs:sequence>  
1595         <xs:element ref="TargetObjectID" />  
1596         <xs:element ref="Subscription" minOccurs="0" />  
1597       </xs:sequence>  
1598     </xs:extension>  
1599   </xs:complexContent>  
1600 </xs:complexType>  
1601  
1602
```

1603 The following is an example of a <GetObjectInfoRequest> message.

```
1604 <GetObjectInfoRequest>  
1605   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>  
1606 </GetObjectInfoRequest>  
1607  
1608
```

1609 3.17.3. GetObjectInfoResponse Message

1610 A PS provider responds to a <GetObjectInfoRequest> message with a <GetObjectInfoResponse> message,
1611 in which the specified Object's information is conveyed.

1612 The <GetObjectInfoResponse> message has the complex type **GetObjectInfoResponseType**, which extends
1613 **ResponseAbstractType** and adds the following elements:

1614 <Object> [**Optional**] The <Object> element is used to convey the information of the requested Object.

1615 The schema declaration for the <GetObjectInfoResponse> message is shown below.

```
1616 <!-- Declaration of GetObjectInfoResponse element -->  
1617 <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>  
1618 <!-- Definition of GetObjectInfoResponseType -->  
1619 <xs:complexType name="GetObjectInfoResponseType">  
1620   <xs:complexContent>  
1621     <xs:extension base="ResponseAbstractType">  
1622       <xs:sequence>  
1623         <xs:element ref="Object" minOccurs="0"/>  
1624       </xs:sequence>  
1625     </xs:extension>  
1626   </xs:complexContent>  
1627 </xs:complexType>  
1628  
1629
```

1630 The following is an example of a <GetObjectInfoResponse> to the <GetObjectInfoRequest> message above.

```
1631 <GetObjectInfoResponse>  
1632   <Status code="OK"/>  
1633   <Object NodeType="urn:liberty:ps:collection">  
1634     <ObjectID>https://ps.com/eiruvoie</ObjectID>  
1635     <DisplayName>Soccer Team</DisplayName>  
1636   </Object>  
1637 </GetObjectInfoResponse>  
1638  
1639
```

1640 3.17.4. Processing Rules

1641 A PS provider:

- 1642 • **MUST** return the <Object> corresponding to the <TargetObjectID> element on the <GetObjectInfoRequest>
1643 message.
- 1644 • **MUST**, if it can not find the targeted Object, respond with *Failed* as the code attribute of the top level
1645 <lu:Status> element. A second level <lu:Status> **MAY** be inserted. If so, the code attribute of that second
1646 level <lu:Status> element **MUST** be set with the following status code:
 - 1647 • **CannotFindObject**
- 1648 • **MUST NOT** respond with any child <Object> and/or <ObjectRef> elements of the specified <Object>.

1649 3.18. Updating Info

1650 A WSC may wish to update or modify the information for an Object, e.g., to change a DisplayName etc.

1651 A WSC uses the <SetObjectInfoRequest> message to update the information for a particular Object. Updateable
1652 information does not include <Object> element, <ObjectRef> element, NodeType attribute, CreatedDateTime
1653 attribute, and ModifiedDateTime attribute.

1654 Note that if a WSC wants to insert a child Object to the particular Object, the WSC MUST use
1655 <AddToCollectionRequest> message (see Section 3.14). Also note that if a WSC wants to remove a child
1656 Object from the particular Object, the WSC MUST use <RemoveFromCollectionRequest> message (see
1657 Section 3.15).

1658 3.18.1. wsa:Action Values

1659 <SetObjectInfoRequest> messages MUST include a <wsa:Action> SOAP header with the value of
1660 "urn:liberty:ps:2006-08:SetObjectInfoRequest." <SetObjectInfoResponse> messages MUST include a
1661 <wsa:Action> SOAP header with the value of "urn:liberty:ps:2006-08:SetObjectInfoResponse."

1662 3.18.2. SetObjectInfoRequest Message

1663 The WSC specifies <Object> elements of the target Object for updating. These <Object> elements MUST
1664 NOT have child <Object> and/or <ObjectRef> elements. Also, these <Object> elements MUST NOT have
1665 CreatedDateTime and ModifiedDateTime attributes.

1666 The <SetObjectInfoRequest> message has the complex type **SetObjectInfoRequestType**, which extends
1667 **RequestAbstractType** and adds the following element:

1668 <Object> [**Required**] The <Object> element is used to convey the updated data of the Object to be updated. The
1669 <Object> element MUST have an <ObjectID> element that identifies the relevant object.

1670 <Subscription> [**Optional**] The <Subscription> element is used to indicate to the PS provider that the WSC
1671 desires that the PS provider send a notification if and when the information of the targeted
1672 Object changes.

1673 The schema declaration for the <SetObjectInfoRequest> message is shown below.

```
1674
1675 <!-- Declaration of SetObjectInfoRequest element -->
1676 <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
1677 <!-- Definition of SetObjectInfoRequestType -->
1678 <xs:complexType name="SetObjectInfoRequestType">
1679   <xs:complexContent>
1680     <xs:extension base="RequestAbstractType">
1681       <xs:sequence>
1682         <xs:element ref="Object" maxOccurs="unbounded"/>
1683         <xs:element ref="Subscription" minOccurs="0"/>
1684       </xs:sequence>
1685     </xs:extension>
1686   </xs:complexContent>
1687 </xs:complexType>
1688
```

1689 The following is an example of a <SetObjectInfoRequest> message in which the WSC is changing the value of
1690 the <DisplayName> element for the specified <Object>. The previously existing value for this element would be
1691 overwritten.

```
1692
1693 <SetObjectInfoRequest>
1694   <Object NodeType="urn:liberty:ps:collection">
```

```
1695         <ObjectID>https://ps.com/eiruvoie</ObjectID>
1696         <DisplayName>Baseball Team</DisplayName>
1697     </Object>
1698 </SetObjectInfoRequest>
1699
```

1700 3.18.3. SetObjectInfoResponse Message

1701 A PS provider responds to a <SetObjectInfoRequest> message with a <SetObjectInfoResponse> message.

1702 The <SetObjectInfoResponse> message has the type of **ResponseAbstractType**

1703 The schema declaration for the <SetObjectInfoResponse> message is shown below.

```
1704
1705 <!-- Declaration of SetObjectInfoResponse element -->
1706 <xs:element name="SetObjectInfoResponse" type="ResponseAbstractType" />
1707
```

1708 The following is an example of a <SetObjectInfoResponse> to the <SetObjectInfoRequest> message above.

```
1709 <SetObjectInfoResponse>
1710     <Status code="OK" />
1711 </SetObjectInfoResponse>
1712
1713
```

1714 3.18.4. Processing Rules

1715 A PS provider:

1716 • **MUST**, if it can not find the target `Object` specified with the `ObjectID`, respond with *Failed* as the code attribute
1717 of the top level <lu:Status> element. A second level <lu:Status> **MAY** be inserted. If so, the code attribute
1718 of that second level <lu:Status> element **MUST** be set with the following status code:

1719 • `CannotFindObject`

1720 • **MUST**, if it finds that the value of the specified `NodeType` attribute is different from the value of the `NodeType`
1721 attribute of the target `Object` specified with the `ObjectID`, respond with *Failed* as the code attribute of the top
1722 level <lu:Status> element. A second level <lu:Status> **MAY** be inserted. If so, the code attribute of that
1723 second level <lu:Status> element **MUST** be set with the following status code:

1724 • `InvalidNodeType`

1725 • **MUST**, if it finds that a WSC specifies a child <Object> element, <ObjectRef> elements, `CreatedDateTime`
1726 attribute, or `ModifiedDateTime` attribute, ignore these elements and/or attributes.

3.19. Querying Objects

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

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

3.19.1. wsa:Action Values

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

3.19.2. QueryObjectsRequest Message

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

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

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

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

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

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

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

```
<!-- Declaration of QueryObjectsRequest element -->
<xs:element name="QueryObjectsRequest" type="QueryObjectsRequestType"/>
<!-- Definition of QueryObjectsRequestType -->
<xs:complexType name="QueryObjectsRequestType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element name="Filter" type="xs:string"/>
        <xs:element ref="Subscription" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
      <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

3.19.2.1. Filter Element

1771 The value of the <Filter> element SHOULD be specified with an XPath expression.

1772 Any XPath expression MUST be evaluated against the set of Objects that would be returned to a hypothetical
1773 <ListMembersRequest> that had targeted the root node and in which the Structured attribute was set to "tree."

1774 For instance, if the WSC wants to get all the <Object> elements with a NodeType attribute with a value of
1775 *urn:liberty:ps:collection* (i.e., the WSC wants to get all the group Objects), the WSC can specify the value of
1776 the <Filter> element as "//ps:Object[@NodeType='urn:liberty:ps:collection']."

1777 The following is an example of a <QueryObjectsRequest> message with which the WSC requests to get all the
1778 <Object> elements that have *urn:liberty:ps:entity* as the value of NodeType element.

```
1779 <QueryObjectsRequest>  
1780   <Filter>//ps:Object[@NodeType='urn:liberty:ps:entity']</Filter>  
1781 </QueryObjectsRequest>  
1782  
1783
```

1784 3.19.2.2. Count and Offset Attributes

1785 When the result of specified criteria has multiple Objects, the WSC may desire to be sent their <Object> elements in
1786 smaller sets (i.e., a smaller number of elements at a time). This is achieved by using the Count and Offset attributes
1787 of the <QueryObjectsRequest> element.

1788 The Count attribute defines how many <Object> elements should be returned in a <QueryObjectsResponse>
1789 message.

1790 The Offset attribute specifies from which element to continue, when requesting for more data. The default value is
1791 zero, which refers to the first <Object> element.

1792 3.19.3. QueryObjectsResponse Message

1793 A PS provider responds to a <QueryObjectsRequest> message with a <QueryObjectsResponse> message.
1794 The <QueryObjectsResponse> contains the <Object> elements that meet the criteria specified in the <Filter>
1795 element of the <QueryObjectsRequest> message.

1796 The <QueryObjectsResponse> message has the complex type **QueryObjectsResponseType**, which extends
1797 **ResponseAbstractType** and adds the following element:

1798 <Object> **[Optional]** The <Object> element is used to convey data of zero or more Objects that the requested
1799 criteria are met to.

1800 The schema declaration for the <QueryObjectsResponse> message is shown below.

```
1801 <!-- Declaration of QueryObjectsResponse element -->
1802 <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType" />
1803 <!-- Definition of QueryObjectsResponseType -->
1804 <xs:complexType name="QueryObjectsResponseType">
1805   <xs:complexContent>
1806     <xs:extension base="ResponseAbstractType">
1807       <xs:sequence>
1808         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded" />
1809       </xs:sequence>
1810     </xs:extension>
1811   </xs:complexContent>
1812 </xs:complexType>
1813 </xs:complexType>
1814
1815
```

1816 The following is an example of a <QueryObjectsResponse> to the <QueryObjectsRequest> message above.

```
1817 <QueryObjectsResponse>
1818   <Status code="OK" />
1819   <Object NodeType="urn:liberty:ps:entity">
1820     <ObjectID>https://ps.com/lstdjfojd</ObjectID>
1821     <DisplayName>Mary</DisplayName>
1822   </Object>
1823   <Object NodeType="urn:liberty:ps:entity">
1824     <ObjectID>https://ps.com/sijfsfsf</ObjectID>
1825     <DisplayName>Bob</DisplayName>
1826   </Object>
1827   <Object NodeType="urn:liberty:ps:entity">
1828     <ObjectID>https://ps.com/reremvls</ObjectID>
1829     <DisplayName>Nick</DisplayName>
1830   </Object>
1831   <Object NodeType="urn:liberty:ps:entity">
1832     <ObjectID>https://ps.com/soijfsfd</ObjectID>
1833     <DisplayName>JoJo</DisplayName>
1834   </Object>
1835   <Object NodeType="urn:liberty:ps:entity">
1836     <ObjectID>https://ps.com/sdosafms</ObjectID>
1837     <DisplayName>Taro</DisplayName>
1838   </Object>
1839   <Object NodeType="urn:liberty:ps:entity">
1840     <ObjectID>https://ps.com/lgsdfsf</ObjectID>
1841     <DisplayName>Hanako</DisplayName>
1842   </Object>
1843 </QueryObjectsResponse>
1844
1845
```

1846 3.19.4. Processing Rules

- 1847 • If a WSC specifies the value of the <Filter> element through an XPath expression, the value SHOULD begin
1848 with the expression "//ps:Object."
- 1849 • All the <Object> elements that are responded in the <QueryObjectsResponse> message from a PS provider
1850 MUST NOT contain any child <Object> and/or <ObjectRef> elements.
- 1851 • If a PS provider cannot find any Objects that meets the criteria that a WSC specifies in the request, the PS provider
1852 MUST respond *OK* as the code attribute of the top level <lu:Status> element. A second level <lu:Status>
1853 MAY be inserted. If so, the code attribute of that second level <lu:Status> element MUST be set with the
1854 following status code:

1893 The schema declaration for the <TestMembershipRequest> message is shown below.

```
1894 <!-- Declaration of TestMembershipRequest element -->
1895 <xs:element name="TestMembershipRequest" type="TestMembershipRequestType"/>
1896 <!-- Definition of TestMembershipRequestType -->
1897 <xs:complexType name="TestMembershipRequestType">
1898   <xs:complexContent>
1899     <xs:extension base="RequestAbstractType">
1900       <xs:sequence>
1901         <xs:element ref="TargetObjectID" minOccurs="0"/>
1902         <xs:element ref="Token"/>
1903         <xs:element ref="Subscription" minOccurs="0"/>
1904       </xs:sequence>
1905     </xs:extension>
1906   </xs:complexContent>
1907 </xs:complexType>
```

1909 The following is an example of a <TestMembershipRequest> message.

```
1910 <TestMembershipRequest>
1911   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
1912   <Token>
1913
1914   </Token>
1915 </TestMembershipRequest>
```

1917 3.20.3. TestMembershipResponse Message

1918 The PS returns the result of the specified membership test in a <Result> element within a
1919 <TestMembershipResponse> message.

1920 The <TestMembershipResponse> message has the complex type **TestMembershipResponseType**, which ex-
1921 tends **ResponseAbstractType** and adds the following elements:

1922 <Result> **[Required]** The <Result> element is used to convey the result of the specified membership test. This
1923 element has a type of **ResultType**, which is derived from **xs:boolean**.

1924 The schema declarations for the <TestMembershipResponse> message and the <Result> element are shown
1925 below.

```
1926 <!-- Definition of ResultType -->
1927 <xs:complexType name="ResultType">
1928   <xs:complexContent>
1929     <xs:extension base="xs:boolean"/>
1930   </xs:complexContent>
1931 </xs:complexType>
1932
1933 <!-- Declaration of TestMembershipResponse element -->
1934 <xs:element name="TestMembershipResponse" type="TestResponseType"/>
1935 <!-- Definition of TestMembershipResponseType -->
1936 <xs:complexType name="TestMembershipResponseType">
1937   <xs:complexContent>
1938     <xs:extension base="ResponseAbstractType">
1939       <xs:sequence>
1940         <xs:element name="Result" type="ResultType" minOccurs="0"/>
1941       </xs:sequence>
1942     </xs:extension>
1943   </xs:complexContent>
1944 </xs:complexType>
```

1946 The following is an example of a `<TestMembershipResponse>` message.

```
1947 <TestMembershipResponse>
1948   <Status code="OK"/>
1949   <Result>true</Result>
1950 </TestMembershipResponse>
```

1952 3.20.4. Processing Rules

1953 If the `<TargetObjectID>` element specifies the `ObjectID` of an `Object` with a `NodeType` attribute of
1954 "urn:liberty:ps:entity," the PS provider **MUST** respond with *Failed* as the code attribute of the top level `<lu:Status>`
1955 element. A second level `<lu:Status>` **MAY** be inserted. If so, the code attribute of that second level `<lu:Status>`
1956 element **MUST** be set with the following status code:

- 1957 • `ObjectIsEntity`

1958 3.21. Resolving Objects

1959 Once a WSC has an `ObjectID` for an entity, it will often desire to communicate with other providers about that user.
1960 To do so, the WSC will first need an identity token for that user. The process of converting an object identifier into an
1961 identity token is referred to as *resolving* the identity token.

1962 3.21.1. wsa:Action Values

1963 `<ResolveIdentifierRequest>` messages **MUST** include a `<wsa:Action>` SOAP header with the value of
1964 "urn:liberty:ps:2006-08:ResolveIdentifierRequest." `<ResolveIdentifierResponse>` messages **MUST** include a
1965 `<wsa:Action>` SOAP header with the value of "urn:liberty:ps:2006-08:ResolveIdentifierResponse."

1966 3.21.2. ResolveIdentifierRequest Message

1967 The WSC can use the `<ResolveIdentifierRequest>` message to ask the PS provider to resolve the specified
1968 `ObjectID` in the `<TargetObjectID>` element, into an appropriate identity token.

1969 An `<ResolveIdentifierRequest>` message consists of one or more `<ResolveInput>` elements. If multiple
1970 `<ResolveInput>` elements are included in a request, then each **MUST** contain a `reqID` attribute so that the response
1971 contents can be correlated to them.

1972 The WSC **MAY** specify its requirements of the identity token through the `<sec:TokenPolicy>`

1973 The `<ResolveIdentifierRequest>` message has the complex type **ResolveIdentifierRequestType**, which
1974 extends **RequestAbstractType** and adds the following elements:

1975 `<ResolveInput>` [**One or more**] The object for which the WSC desires an identity token be resolved.

1976 The schema declaration for the <ResolveIdentifierRequest> message is shown below.

```
1977
1978 <!-- Declaration of ResolveIdentifierRequest element -->
1979 <xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType" />
1980 <!-- Definition of ResolveIdentifierRequestType -->
1981 <xs:complexType name="ResolveIdentifierRequestType">
1982   <xs:complexContent>
1983     <xs:extension base="RequestAbstractType">
1984       <xs:sequence>
1985         <xs:element ref="ResolveInput" maxOccurs="unbounded" />
1986       </xs:sequence>
1987     </xs:extension>
1988   </xs:complexContent>
1989 </xs:complexType>
1990
```

1991 3.21.2.1. ResolveInput element

1992 The <ResolveInput> element is of complex type <ResolveInputType>, which extends the complex type
1993 <MappingInputType> defined in [[LibertyAuthn](#)] to add the following element:

1994 <TargetObjectID> **[Required]** The <TargetObjectID> conveys the ObjectID of the target entity Object for
1995 which the WSC is requesting an identity token be resolved.

1996 The schema declaration for the <ResolveInput> element is shown below.

```
1997
1998 <!-- Declaration of ResolveInput element -->
1999 <xs:element name="ResolveInput" type="ResolveInputType" />
2000 <!-- Definition of ResolveInputType -->
2001 <xs:complexType name="ResolveInputType">
2002   <xs:complexContent>
2003     <xs:extension base="ims:MappingInputType">
2004       <xs:sequence>
2005         <xs:element ref="TargetObjectID" />
2006       </xs:sequence>
2007     </xs:extension>
2008   </xs:complexContent>
2009 </xs:complexType>
2010
```

2011 The semantics and processing rules of the elements and attributes are as defined in [[LibertyAuthn](#)].

2012 The following is an example of a <ResolveIdentifierRequest> message in which the WSC is requesting that an
2013 identity token for the Object identified by the specified ObjectID be returned. The WSC is indicating that it desires
2014 a transient identifier for the identity token.

```
2015
2016 <ResolveIdentifierRequest>
2017   <ResolveInput>
2018     <TargetObjectID>https://ps.com/lgsdfsf/</TargetObjectID>
2019     <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
2020       <samlp2:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
2021         SPNameQualifier="https://psa.com" />
2022     </sec:TokenPolicy>
2023   </ResolveInput>
2024 </ResolveIdentifierRequest>
2025
```

2026 3.21.3. ResolveIdentifierResponse Message

2027 A PS provider responds to a <ResolveIdentifierRequest> message with a <ResolveIdentifierResponse>
2028 message. The PS provider returns the identity tokens corresponding to the Objects specified by the ObjectID in the
2029 TargetObjectID element on the <ResolveIdentifierRequest> message.

2030 The schema declaration for the <ResolveIdentifierResponse> message is shown below.

```
2031 <!-- Declaration of ResolveIdentifierResponse element -->
2032 <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType" />
2033 <!-- Definition of ResolveIdentifierResponseType -->
2034 <xs:complexType name="ResolveIdentifierResponseType">
2035   <xs:complexContent>
2036     <xs:extension base="ResponseAbstractType">
2037       <xs:sequence>
2038         <xs:element ref="ResolveOutput" maxOccurs="unbounded" />
2039       </xs:sequence>
2040     </xs:extension>
2041   </xs:complexContent>
2042 </xs:complexType>
2043
2044
```

2045 3.21.3.1. ResolveOutput element

2046 Each <ResolveOutput> element consists of an identity token (in the form of a <sec:Token>).

2047 The element is of complex type MappingOutputType defined in [[LibertyAuthn](#)]. Returned tokens are correlated with
2048 input object identifiers through the reference mechanism defined in [[LibertyAuthn](#)]

2049 The declaration for the <ResolveOutput> element is shown below.

```
2050 <!-- Declaration of ResolveOutput element -->
2051 <xs:element name="ResolveOutput" type="ims:MappingOutputType" />
2052
2053
```

2054 The following is an example of a <ResolveIdentifierResponse> to the <ResolveIdentifierRequest>
2055 message above.

```
2056 <ResolveIdentifierResponse>
2057   <Status code="OK" />
2058   <ResolveOutput>
2059     <Token>
2060
2061   </Token>
2062 </ResolveOutput>
2063 </ResolveIdentifierResponse>
2064
```

2065 3.21.4. Processing Rules

2066 • The PS provider MUST, if unable to find one or more (but not all) specified input objects, respond "PartialSuccess"
2067 as the code attribute of the top level <lu:Status> element, and MAY use second level <lu:Status> elements
2068 that containing a ref attribute equal to the associated <ResolveInput>'s reqID attribute. If unable to find any input
2069 objects, the PS provider MUST respond "Failed" as the code attribute of the top level <lu:Status> element. A
2070 second level <lu:Status> MAY be inserted. If so, the code attribute of that second level <lu:Status> element
2071 MUST be set with the following status code:

2072 • CannotFindObject

2073 • Upon receiving a `<ResolveIdentifierRequest>` from an SP carrying `<TargetObjectID>` elements that
2074 correspond to an existing `Object`, the PS provider MUST endeavor to return to the SP an appropriate identity
2075 token corresponding to that `Object`.

2076 To do so, the PS provider MAY, itself, send an `<ims:IdentityMappingRequest>` message to the relevant
2077 identity provider for the `Object` in question, specifying the requesting WSC as the target namespace.

2078 • If a PS provider cannot resolve any of the input objects into identity tokens, the PS provider MUST re-
2079 spond "Failed" as the code attribute of the top level `<lu:Status>` element, and SHOULD use second level
2080 `<lu:Status>` elements that MUST contain a `ref` attribute equal to the associated `<ResolveInput>`'s `reqID` at-
2081 tribute. The code attribute of any second level `<lu:Status>` element MUST be set with the following status
2082 code:

2083 • `CannotResolveToken`

2084 • If a PS provider is unable to resolve all input objects into identity tokens, but is able to resolve some, the
2085 PS provider MUST respond "PartialSuccess" as the code attribute of the top level `<lu:Status>` element, and
2086 SHOULD use second level `<lu:Status>` elements that MUST contain a `ref` attribute equal to the associated
2087 `<ResolveInput>`'s `reqID` attribute. The code attribute of any second level `<lu:Status>` element MUST be set
2088 with the following status code:

2089 • `CannotResolveToken`

2090 • The PS provider MUST, if one or more (but not all) specified input objects is of type "collection," respond
2091 "PartialSuccess" as the code attribute of the top level `<lu:Status>` element, and SHOULD use second level
2092 `<lu:Status>` elements that contain a `ref` attribute equal to the associated `<ResolveInput>`'s `reqID` attribute.
2093 If all input objects are of type "collection," the PS provider MUST respond "Failed" as the code attribute of the
2094 top level `<lu:Status>` element. The second level `<lu:Status>` elements corresponding to such failed inputs
2095 MUST be set with the following status code:

2096 • `ObjectIsCollection`

4. Interaction with Users

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

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

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

4.1. Model (Informative)

The invitation model is as follows:

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

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

4.2. Additional Federations for Sharing of Identity Services

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

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

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

2135 To facilitate this operation, the DS of the inviting user will likely need to provide an appropriate WSP
2136 <EndpointReference> to the WSC upon that WSC querying for relevant (associated with the inviting user
2137 and of a particular service type) WSPs. If the inviting user's policy is such that merely the fact of existence or
2138 location of their identity data (beyond its actual value) warrants protection, then the DS should apply access control
2139 mechanisms to such WSC queries. To do so, it may need to have an identifier for the requesting user so that an
2140 appropriate access control decision can be made.

2141 In this sense, the existence and location of a user's identity services (as exemplified in the endpoint references pointing
2142 there), are identity resources like any other identity service and may require appropriate federations be established to
2143 enable access control.

2144 If policy is such that no such fine-grained access control for <EndpointReference>s is deemed necessary, then
2145 establishing a federation between the DS of the inviting user and the IDP of the invited user will not be necessary.

2146 **4.3. Consent Model**

2147 A number of consent models governing the various federations established through the invitation process are possible.
2148 A few examples, amongst others, are listed below:

2149 One possibility would be for an invited user, when federating their IDP to the PS provider of a friend upon an invitation
2150 from that friend, to specify that the establishment of subsequent federations between service providers and their IDP
2151 need not require additional consent. Such blanket consent, while optimizing usability, could present unforeseen risks,
2152 e.g., giving a user the permission, even if never taken advantage of, to view offensive online material.

2153 At the other extreme, a user could request that they be asked for specific consent for each and every such federation.

2154 Additionally, a user could specify that their IDP be allowed to establish "provisional" federated identifiers with other
2155 SPs, but that they expect to be given the opportunity to give explicit consent to these federations if and when they
2156 attempt to use them. The identifiers are provisional in the sense that the IDP will not actually agree to use them with
2157 the other provider until such consent is obtained, at which point they would become "real." This model would ensure
2158 that the user would not be presented with numerous requests for consent - consent would be obtained only when
2159 necessary.

2160 **4.4. Elements Supporting Invitation**

2161 The following elements support the invitation model. These elements are optional within the appropriate protocol
2162 messages.

2163 **4.4.1. PStoSPRedirectURL element**

2164 The SP MAY use the <PStoSPRedirectURL> element on <AddEntityRequest> and <AddKnownEntityRequest>
2165 messages to specify to the PS provider the URL (at the SP) to which that SP desires the invited user's user agent be
2166 directed after successful interaction and IDP federation has occurred. If and when the invited user's user agent has
2167 been sent to this URL, the SP MAY provide the invited user the designated access to the resource in question.

2168 The value of the <PStoSPRedirectURL> element MUST be such that, if and when a user agent is sent to this address
2169 from the PS provider, the SP can unambiguously determine the invitation to which the URL corresponds. The URL
2170 MUST be unique for each combination of the inviting user, the PS, and the invited user.

2171 **4.4.2. <SptoPSRedirectURL> element**

2172 The PS provider MAY use the <SptoPSRedirectURL> element on <AddEntityResponse> and
2173 <AddKnownEntityResponse> messages to specify to the SP the URL (at the PS provider) to which that PS
2174 desires the invited user's user agent be directed after successful initial interaction at the SP has occurred. If and when

2175 the invited user's user agent is sent to this URL, the PS provider will endeavour to establish a federation for that
2176 principal with the appropriate IDP.

2177 The value of the <SPToPSRedirectURL> element MUST be such that, if and when a user agent is sent to this address
2178 from the SP, the PS can unambiguously determine the invitation to which the URL corresponds. The URL MUST be
2179 unique for each combination of the inviting user (that owns the PS resource), the requesting SP, and the invited user.

2180 The PS provider MUST be prepared for the invited user to, at some point in the future, visit the URL provided in
2181 any specified <SPToPSRedirectURL> element. As it may be some time before the invited user does respond, the PS
2182 provider SHOULD store this url for a reasonable length of time.

2183 4.4.2.1. Processing Rules

2184 If and when the invited user responds to the invitation, the SP:

2185 • MUST, after appropriately informing and "consenting" the invited user, direct the user agent to the address
2186 previously specified within the <SPToPSRedirectURL> element.

2187 Once the invited user has been redirected to the PS provider, the PS provider:

2188 • MUST determine the invited user's IDP (or preferred IDP if the invited user has multiple).

2189 MUST endeavor to establish a federated identity for that user with that IDP.

2190 • MAY obtain an identity token from the IDP for the invited user targeted for itself.

2191 • MUST redirect the invited user's agent to the address previously specified by any <PStoSPRedirectURL>
2192 element in the original <AddEntityRequest> or <AddKnownEntityRequest> message.

2193 Once the PS has redirected the invited user to the <PStoSPRedirectURL> address, the SP MAY choose to send
2194 a <samlp:AuthnRequest> message to the IDP asking for a <saml:AuthnStatement> attesting to that user's
2195 authentication status there.

2196 In its corresponding <samlp:Response>, the IDP SHOULD use the same subject identifier for this
2197 <saml:Assertion> as previously delivered to the SP within the identity token through the PS provider and
2198 the <Notify> message (unless any SP policy on the <samlp:AuthnRequest> message precludes this).

2199 4.4.3. <QueryString> element

2200 The <QueryString> element enables an alternative model for invited user interaction which is expected to better
2201 defend against identity theft attacks in which a valid email is spoofed to fool users into clicking an embedded
2202 URL. The invitation received by the invited user will contain a string carrying a SAML artifact (and potentially
2203 relay state info) representing a SAML <samlp:AuthnRequest> message created by the PS provider. The invited
2204 user can, if they choose, present this artifact string to their identity provider - which can then use the SAML
2205 <samlp:ArtifactResolve> message to retrieve the original <samlp:AuthnRequest> message from the PS
2206 provider.

2207 As the invited user visits their IDP by explicitly providing the address or using an existing bookmark, they can be more
2208 confident that the site is not spoofed. Once they are at their IDP and after presenting the SAML artifact, appropriate
2209 federations can be established for the invited user with the originating PS provider and SP.

2210 4.4.3.1. Schema

```
2211 <xs:element name="QueryString" type="QueryStringType" />
2212   <xs:complexType name="QueryStringType">
2213     <xs:simpleContent>
2214       <xs:extension base="xs:string" />
2215     </xs:simpleContent>
```

2216 </xs:complexType>
2217

2218 4.4.3.2. Formatting Rules

2219 The contents of the <QueryString> element MUST satisfy the formatting requirements of the URL encoding of the
2220 SAML Artifact Binding (see [[SAMLBind2](#)]) as identified by the URI:

2221 *urn:oasis:names:tc:SAML:2.0:artifact-04.*

2222 The following is an example of an <QueryString> element in which the PS Provider has included relay state
2223 information through an additional RelayState parameter.

2224
2225 <QueryString>SAMLart=AAQAADWNEw5VT47wcO4zX%2FiEzMmFQv%3D&RelayState=0043bfc1bc45110dae170
2226 04005b13a2b</QueryString>

2227 The contents of the above element would be communicated to the invited user by the SP (either directly or indirectly
2228 through the inviting user) for them to provide to their IDP.

2229 4.4.3.3. Processing Rules

2230 To support this invitation model, when responding to either a <AddEntityRequest> or <AddKnownEntityRequest>
2231 message, the PS provider:

- 2232 • MUST create an artifact string in accordance with the SAML Artifact Binding (see [[SAMLBind2](#)]).
- 2233 • MUST insert this string within an <QueryString> element in the <AddEntityResponse> or
2234 <AddKnownEntityResponse> message.
- 2235 • MUST be prepared for, at some point in the future, an IDP to send an <samlp:ArtifactResolve> message as a
2236 consequence of the invited user presenting the contents of any specified <QueryString> element to the IDP. As
2237 it may be some time before the invited user does present the artifact to their IDP, the PS provider SHOULD store
2238 the artifact for a reasonable length of time.
- 2239 MUST return the appropriate <samlp:AuthnRequest> message in its <samlp:ArtifactResponse> message.
- 2240 MAY, when the IDP returns a name identifier (either pre-existing or generated) for the invited user in its
2241 <samlp:Response> message, send an <ims:IdentityMappingRequest> message to the IDP requesting an
2242 identity token (targeted at itself) for the invited user unless it already has such a identity token.

2243 After receiving an <AddEntityResponse> or <AddKnownEntityResponse> message with an <QueryString>
2244 element, the SP:

- 2245 • MUST extract the contents of the <QueryString> element
- 2246 • MUST attempt to communicate the extracted string to the invited user.

2247 If and when the invited user presents the query string that it received from the SP to its IDP, the IDP:

- 2248 • MUST authenticate the user
- 2249 • MUST determine the identity of the PS provider from the artifact string *SourceID* and determine the addresses to
2250 which <samlp:ArtifactResolve> and <Response> messages are to be sent (e.g., through metadata).
- 2251 • MUST send an <samlp:ArtifactResolve> message to the PS provider and MUST process the retrieved
2252 <samlp:AuthnRequest> message in accordance with the SSO Profiles of SAML [[SAMLProf2](#)].

- 2253 • MUST create and deliver a `<samlp:Response>` message to the PS provider using a SAML front-channel binding
2254 (e.g., HTTP Redirect, HTTP POST, or HTTP Artifact).
- 2255 If the value of the presented query string included any relay state information, the binding by which the
2256 `<samlp:Response>` message is delivered to the PS MUST support the communication of this relay state
2257 information back to the PS provider.
- 2258 Once the invited user has been redirected to the PS provider and the PS provider has obtained the `<samlp:Response>`,
2259 the PS provider:
- 2260 • MUST extract the name identifier from within the `<Subject>` of the `<Assertion>`.
- 2261 • MUST determine the original `<AddEntityRequest>` or `<AddKnownEntityRequest>` message to which the
2262 returned identifier corresponds.
- 2263 • MUST use the appropriate federated name identifier for the user to obtain an identity token from the IDP for the
2264 invited user - this identity token targeted for itself.
- 2265 • MUST, unless the SP did not include a `<Subscription>` in its `<AddEntityRequest>` message, obtain an
2266 identity token from the IDP for the invited user targeted at the SP.
- 2267 MUST forward on the identity token just received from the IDP in a `<Notify>` message, specifying the
2268 `SubscriptionID` of the previous `<Subscription>` element.
- 2269 • MUST redirect the invited user's agent to the address previously specified by the `<PStoSPRedirectURL>` element
2270 in the original `<AddEntityRequest>` or `<AddKnownEntityRequest>` message.
- 2271 Once the invited user has been redirected to the `<PStoSPRedirectURL>` address, the SP:
- 2272 • MAY choose to send a `<samlp:AuthnRequest>` message to the IDP asking for a `<saml:AuthnStatement>`
2273 attesting to that user's authentication status there. In its `<Response>`, the IDP MUST use the same subject
2274 identifier for this `<saml:Assertion>` as previously delivered to the SP within the identity token through the
2275 PS provider and the `<Notify>` message.

5. Sequence Examples

Following are detailed sequence examples for:

- setting access control against a PS group
- checking group membership for access control
- performing a collective operation against group members

5.1. Policy definition

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

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

```
<ListMembersRequest Structured="children" />
```

4. PS responds with the Objects.

```
<ListMembersResponse>  
  <Status code="OK" />  
  <Object NodeType="urn:liberty:ps:entity" >  
    <ObjectID>https://psa.com/sdfhgusfsf</ObjectID>  
    <DisplayName>Bob</DisplayName>  
  </Object>  
  <Object NodeType="urn:liberty:ps:entity">  
    <ObjectID>https://psa.com/itndojd</ObjectID>  
    <DisplayName>Mary</DisplayName>  
  </Object>  
  <Object NodeType="urn:liberty:ps:collection">  
    <ObjectID>https://psa.com/sijfsfsf</ObjectID>  
    <DisplayName>Work Friends</DisplayName>  
  </Object>  
  <Object NodeType="urn:liberty:ps:collection">  
    <ObjectID>https://psa.com/lsdjfojd</ObjectID>  
    <DisplayName>Soccer Team</DisplayName>  
  </Object>  
</ListMembersResponse>
```

5. SPs displays the list to Alice.

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

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

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

5.2. AccessControl

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

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

2. SPa asks "Who are you?"

3. Bob says "Ask IDPb."

4. SPa redirects Bob to IDPb with AuthnRequest.

```
<samlp:AuthnRequest
  ID="NTT7630E00861279F0ADC63E241D0926D0B"
  Version="2.0" IssueInstant="...">
  <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
    https://spa.com
  </saml:Issuer>
  <samlp:NameIDPolicy
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
</samlp:AuthnRequest>
```

5. IDPb authenticates Bob.

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

```
<samlp:Response
  ID="NTT3F633E3F712BAC4B0804714431D46D7B"
  InResponseTo="NTT7630E00861279F0ADC63E241D0926D0B"
  Version="2.0" IssueInstant="...">
  <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
    https://idpb.com
  </saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
  <saml:Assertion
    Version="2.0" IssueInstant="..."
    ID="NTT02062BBDE3E97EF0749828BCB8C15DFB">
    <saml:Issuer
      Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
      https://idpb.com
    </saml:Issuer>
    <saml:Subject>
      <saml:NameID
        NameQualifier="https://idpb.com"
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
        e0b735bf9d1f3959241d3584733d704c
      </saml:NameID>
    </saml:Subject>
    <saml:AuthnStatement
      AuthnInstant="..." SessionIndex="..."
      <saml:AuthnContext>AuthnContext goes here</saml:AuthnContext>
    </saml:AuthnStatement>
  </saml:Assertion>
</samlp:Response>
```

2376 7. SPa sends IDPb an <ims:IdentityMappingRequest>, providing the previous name identifier for Bob and
2377 specifying PSa as the target namespace.

```
2378 <soap:Envelope>
2379 <soap:Header>
2380   <ws:Security>
2381     <saml:Assertion ID="assertionid">
2382       credentials for Bob at IDPb
2383     </saml:Assertion>
2384   </ws:Security>
2385 </soap:Header>
2386 <soap:Body>
2387 <ims:IdentityMappingRequest>
2388 <ims:MappingInput>
2389 <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
2390   <samlp:NameIDPolicy SPNameQualifier="https://psa.com"
2391     Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent" />
2392 </sec:TokenPolicy>
2393 <sec:Token ref="#assertionid"/>
2394 </ims:MappingInput>
2395 </ims:IdentityMappingRequest>
2396 </soap:Body>
2397 </soap:Envelope>
2398
2399
```

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

```
2401 <ims:IdentityMappingResponse>
2402 <ims:MappingOutput>
2403 <sec:Token>
2404   <saml:Assertion>
2405     identity token for Bob in PSa's namespace
2406   </saml:Assertion>
2407 </sec:Token>
2408 </ims:MappingOutput>
2409 </ims:IdentityMappingResponse>
2410
2411
```

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 <TestMembershipRequest>
2415 <TargetObjectID>https://ps.com/sijfsfs</TargetObjectID>
2416 <sec:Token>
2417   <saml:Assertion>
2418     identity token for Bob in PSa's namespace
2419   </saml:Assertion>
2420 </sec:Token>
2421 </TestMembershipRequest>
2422
2423
```

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

2433 12. Confident that Bob is a member of the group against which Alice defined privileges, SPa grants Bob access to the
2434 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 <ListMembersRequest>  
2444   <TargetObjectID>https://ps.com/nmerflas</TargetObjectID>  
2445 </ListMembersRequest>  
2447
```

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

```
2449 <ListMembersResponse>  
2450   <Status code="OK"/>  
2451   <Object NodeType="urn:liberty:ps:entity" >  
2452     <ObjectID>https://psa.com/sdfhgusfsf</ObjectID>  
2453     <DisplayName>Bob</DisplayName>  
2454   </Object>  
2455   <Object NodeType="urn:liberty:ps:entity" >  
2456     <ObjectID>https://psa.com/itndojd</ObjectID>  
2457     <DisplayName>Mary</DisplayName>  
2458   </Object>  
2459 </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 <ResolveIdentifierRequest>  
2467   <ResolveInput reqID="0">  
2468     <TargetObjectID>https://psa.com/sdfhgusfsf</TargetObjectID> <!-- Bob -->  
2469   </ResolveInput>  
2470   <ResolveInput reqID="1">  
2471     <TargetObjectID>https://psa.com/itndojd</TargetObjectID> <!-- Alice -->  
2472   </ResolveInput>  
2473 </ResolveIdentifierRequest>  
2474  
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 <ims:IdentityMappingRequest>
2479   <ims:MappingInput reqID="2">
2480     <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
2481       <samlp:NameIDPolicy SPNameQualifier="https://spa.com"
2482         Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
2483     </sec:TokenPolicy>
2484     <sec:Token>
2485       <saml:Assertion>
2486         existing identity token for Bob between PSa and IDPb
2487       </saml:Assertion>
2488     </sec:Token>
2489   </ims:MappingInput>
2490   <ims:MappingInput reqID="3">
2491     <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
2492       <samlp:NameIDPolicy SPNameQualifier="https://spa.com"
2493         Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
2494     </sec:TokenPolicy>
2495     <sec:Token>
2496       <saml:Assertion>
2497         existing identity token for Alice between PSa and IDPb
2498       </saml:Assertion>
2499     </sec:Token>
2500   </ims:MappingInput>
2501 </ims:IdentityMappingRequest>
2502
2503
```

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

```
2506 <ims:IdentityMappingResponse>
2507   <ims:MappingOutput reqRef="2">
2508     <Status>OK</Status>
2509     <sec:Token>
2510       <saml:Assertion>
2511         an identity token for Bob in SPa/WSCa's namespace goes here
2512       </saml:Assertion>
2513     </sec:Token>
2514   </ims:MappingOutput>
2515   <ims:MappingOutput reqRef="3">
2516     <Status>OK</Status>
2517     <sec:Token>
2518       <saml:Assertion>
2519         an identity token for Alice in SPa/WSCa's namespace goes here
2520       </saml:Assertion>
2521     </sec:Token>
2522   </ims:MappingOutput>
2523 </ims:IdentityResponse>
2524
2525
```

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 <ResolveIdentifierResponse>
2529   <Status code="OK"/>
2530   <ResolveOutput reqRef="0">
2531     <Token>
2532       <saml:Assertion>
2533         an identity token for Bob in SPa/WSCa's namespace goes here
2534       </saml:Assertion>
2535     </Token>
2536
```

```
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](#)].

7. XML Schema for ID-WSF People Service

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

```
2562 <xs:schema
2563   targetNamespace="urn:liberty:ps:2006-08"
2564   xmlns="urn:liberty:ps:2006-08"
2565   xmlns:lu="urn:liberty:util:2006-08"
2566   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2567   xmlns:ims="urn:liberty:ims:2006-08"
2568   xmlns:subs="urn:liberty:ssos:2006-08"
2569   xmlns:sec="urn:liberty:security:2006-08"
2570   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
2571   elementFormDefault="qualified"
2572   attributeFormDefault="unqualified">
2573
2574   <xs:import namespace="urn:liberty:util:2006-08"
2575     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
2576   <xs:import namespace="urn:liberty:ims:2006-08"
2577     schemaLocation="liberty-idwsf-idmapping-svc-v2.0.xsd"/>
2578   <xs:import namespace="urn:liberty:ssos:2006-08"
2579     schemaLocation="liberty-idwsf-subsv1.0.xsd"/>
2580   <xs:import namespace="urn:liberty:security:2006-08"
2581     schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>
2582   <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
2583     schemaLocation="saml-schema-protocol-2.0.xsd"/>
2584
2585   <!-- Definition of LocalizedDisplayNameType -->
2586   <xs:complexType name="LocalizedDisplayNameType">
2587     <xs:simpleContent>
2588       <xs:extension base="xs:string">
2589         <xs:attribute name="Locale" type="xs:language" use="optional"/>
2590         <xs:attribute name="IsDefault" type="xs:boolean" use="optional"/>
2591       </xs:extension>
2592     </xs:simpleContent>
2593   </xs:complexType>
2594
2595   <!-- Definition of TagType -->
2596   <xs:complexType name="TagType">
2597     <xs:simpleContent>
2598       <xs:extension base="xs:string">
2599         <xs:attribute name="Ref" type="xs:anyURI" use="required"/>
2600       </xs:extension>
2601     </xs:simpleContent>
2602   </xs:complexType>
2603
2604   <!-- Declaration of ObjectID element -->
2605   <xs:element name="ObjectID" type="ObjectIDType"/>
2606
2607   <!-- Declaration of TargetObjectID element -->
2608   <xs:element name="TargetObjectID" type="ObjectIDType"/>
2609
2610   <!-- Definition of ObjectIDType -->
2611   <xs:complexType name="ObjectIDType">
2612     <xs:simpleContent>
2613       <xs:extension base="xs:anyURI"/>
2614     </xs:simpleContent>
2615   </xs:complexType>
2616
2617   <!-- Declaration of Object element -->
2618   <xs:element name="Object" type="ObjectType"/>
2619
2620   <!-- Definition of ObjectType -->
2621   <xs:complexType name="ObjectType">
2622     <xs:sequence>
2623       <xs:element ref="ObjectID" minOccurs="0"/>
2624     </xs:sequence>
2625   </xs:complexType>
```

```

2625     <xs:element name="DisplayName" type="LocalizedDisplayNameType"
2626         minOccurs="1" maxOccurs="unbounded"/>
2627     <xs:element name="Tag" type="TagType" minOccurs="0" maxOccurs="unbounded"/>
2628     <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2629     <xs:element name="ObjectRef" type="ObjectIDType" minOccurs="0" maxOccurs="unbounded"/>
2630 </xs:sequence>
2631 <xs:attribute name="NodeType" type="xs:anyURI" use="required"/>
2632 <xs:attribute name="CreatedDateTime" type="xs:dateTime" use="optional"/>
2633 <xs:attribute name="ModifiedDateTime" type="xs:dateTime" use="optional"/>
2634 </xs:complexType>
2635
2636 <!-- Declaration of PStoSPRedirectURL-->
2637
2638 <xs:element name="PStoSPRedirectURL" type="PStoSPRedirectURLType"/>
2639
2640 <!-- Definition of PStoSPRedirectURLType-->
2641
2642 <xs:complexType name="PStoSPRedirectURLType">
2643     <xs:annotation>
2644         <xs:documentation>
2645             When sending a AddEntityRequest to a PS provider,
2646             the SP may insert a PStoSPRedirectURL. It will be
2647             to this URL that the invited principals will be
2648             sent after federating their IDP account to the PS
2649             provider.
2650         </xs:documentation>
2651     </xs:annotation>
2652     <xs:simpleContent>
2653         <xs:extension base="xs:anyURI"/>
2654     </xs:simpleContent>
2655 </xs:complexType>
2656
2657 <!-- Declaration of SPToPSRedirectURL-->
2658
2659 <xs:element name="SPToPSRedirectURL" type="SPToPSRedirectURLType"/>
2660
2661 <!-- Definition of SPToPSRedirectURLType-->
2662
2663 <xs:complexType name="SPToPSRedirectURLType">
2664     <xs:annotation>
2665         <xs:documentation>
2666             A PS provider may insert a SPToPSRedirectURL in its
2667             AddEntityResponse. It will be to this URL that the
2668             invited principal will be sent after responding to the
2669             invitation.
2670         </xs:documentation>
2671     </xs:annotation>
2672     <xs:simpleContent>
2673         <xs:extension base="xs:anyURI"/>
2674     </xs:simpleContent>
2675 </xs:complexType>
2676
2677 <!-- Declaration of QueryString -->
2678
2679 <xs:element name="QueryString" type="QueryStringType"/>
2680
2681 <!-- Definition of QueryStringType-->
2682
2683 <xs:complexType name="QueryStringType">
2684     <xs:annotation>
2685         <xs:documentation>
2686             A PS provider may insert a QueryString in its
2687             AddEntityResponse or AddKnownEntityResponse. The
2688             invited Principal can present this artifact string
2689             to a certain provider.
2690         </xs:documentation>
2691     </xs:annotation>

```

```

2692     <xs:simpleContent>
2693         <xs:extension base="xs:string" />
2694     </xs:simpleContent>
2695 </xs:complexType>
2696
2697 <!-- Declaration of CreatePSObject element -->
2698 <xs:element name="CreatePSObject" />
2699
2700 <!-- Definition of RequestAbstractType -->
2701 <xs:complexType name="RequestAbstractType" abstract="true">
2702     <xs:anyAttribute namespace="##other" processContents="lax" />
2703 </xs:complexType>
2704
2705 <!-- Definition of ResponseAbstractType -->
2706 <xs:complexType name="ResponseAbstractType" abstract="true">
2707     <xs:sequence>
2708         <xs:element ref="lu:Status" />
2709     </xs:sequence>
2710     <xs:anyAttribute namespace="##other" processContents="lax" />
2711 </xs:complexType>
2712
2713 <!-- Declaration of AddEntityRequest element -->
2714 <xs:element name="AddEntityRequest" type="AddEntityRequestType" />
2715 <!-- Definition of AddEntityRequestType -->
2716 <xs:complexType name="AddEntityRequestType">
2717     <xs:complexContent>
2718         <xs:extension base="RequestAbstractType">
2719             <xs:sequence>
2720                 <xs:element ref="Object" />
2721                 <xs:element ref="PStoSPRedirectURL" minOccurs="0" />
2722                 <xs:element ref="CreatePSObject" minOccurs="0" />
2723                 <xs:element ref="Subscription" minOccurs="0" />
2724                 <xs:element ref="sec:TokenPolicy" minOccurs="0" />
2725             </xs:sequence>
2726         </xs:extension>
2727     </xs:complexContent>
2728 </xs:complexType>
2729
2730 <!-- Declaration of AddEntityResponse element -->
2731 <xs:element name="AddEntityResponse" type="AddEntityResponseType" />
2732 <!-- Definition of AddEntityResponseType -->
2733 <xs:complexType name="AddEntityResponseType">
2734     <xs:complexContent>
2735         <xs:extension base="ResponseAbstractType">
2736             <xs:sequence>
2737                 <xs:element ref="Object" minOccurs="0" />
2738                 <xs:element ref="SPtoPSRedirectURL" minOccurs="0" maxOccurs="1" />
2739                 <xs:element ref="QueryString" minOccurs="0" maxOccurs="1" />
2740             </xs:sequence>
2741         </xs:extension>
2742     </xs:complexContent>
2743 </xs:complexType>
2744
2745 <!-- Declaration of AddKnownEntityRequest element -->
2746 <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType" />
2747 <!-- Definition of AddKnownEntityRequestType -->
2748 <xs:complexType name="AddKnownEntityRequestType">
2749     <xs:complexContent>
2750         <xs:extension base="RequestAbstractType">
2751             <xs:sequence>
2752                 <xs:element ref="Object" />
2753                 <xs:element ref="sec:Token" />
2754                 <xs:element ref="CreatePSObject" minOccurs="0" />
2755                 <xs:element ref="Subscription" minOccurs="0" />
2756                 <xs:element ref="sec:TokenPolicy" minOccurs="0" />
2757             </xs:sequence>
2758         </xs:extension>

```

```

2759     </xs:complexContent>
2760 </xs:complexType>
2761
2762 <!-- Declaration of AddKnownEntityResponse element -->
2763 <xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType" />
2764 <!-- Definition of AddKnownEntityResponseType -->
2765 <xs:complexType name="AddKnownEntityResponseType">
2766   <xs:complexContent>
2767     <xs:extension base="ResponseAbstractType">
2768       <xs:sequence>
2769         <xs:element ref="Object" minOccurs="0" />
2770         <xs:element ref="SptoPSRedirectURL" minOccurs="0" maxOccurs="1" />
2771         <xs:element ref="QueryString" minOccurs="0" maxOccurs="1" />
2772       </xs:sequence>
2773     </xs:extension>
2774   </xs:complexContent>
2775 </xs:complexType>
2776
2777 <!-- Declaration of AddCollectionRequest element -->
2778 <xs:element name="AddCollectionRequest" type="AddCollectionRequestType" />
2779 <!-- Definition of AddCollectionRequestType -->
2780 <xs:complexType name="AddCollectionRequestType">
2781   <xs:complexContent>
2782     <xs:extension base="RequestAbstractType">
2783       <xs:sequence>
2784         <xs:element ref="Object" />
2785         <xs:element ref="Subscription" minOccurs="0" />
2786       </xs:sequence>
2787     </xs:extension>
2788   </xs:complexContent>
2789 </xs:complexType>
2790
2791 <!-- Declaration of AddCollectionResponse element -->
2792 <xs:element name="AddCollectionResponse" type="AddCollectionResponseType" />
2793 <!-- Definition of AddCollectionResponseType -->
2794 <xs:complexType name="AddCollectionResponseType">
2795   <xs:complexContent>
2796     <xs:extension base="ResponseAbstractType">
2797       <xs:sequence>
2798         <xs:element ref="Object" minOccurs="0" />
2799       </xs:sequence>
2800     </xs:extension>
2801   </xs:complexContent>
2802 </xs:complexType>
2803
2804 <!-- Declaration of AddToCollectionRequest element -->
2805 <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType" />
2806 <!-- Definition of AddToCollectionRequestType -->
2807 <xs:complexType name="AddToCollectionRequestType">
2808   <xs:complexContent>
2809     <xs:extension base="RequestAbstractType">
2810       <xs:sequence>
2811         <xs:element ref="TargetObjectID" />
2812         <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded" />
2813         <xs:element ref="Subscription" minOccurs="0" />
2814       </xs:sequence>
2815     </xs:extension>
2816   </xs:complexContent>
2817 </xs:complexType>
2818
2819 <!-- Declaration of AddToCollectionResponse element -->
2820 <xs:element name="AddToCollectionResponse" type="ResponseAbstractType" />
2821
2822 <!-- Declaration of RemoveEntityRequest element -->
2823 <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType" />
2824 <!-- Definition of RemoveEntityRequestType -->
2825 <xs:complexType name="RemoveEntityRequestType">

```

```

2826     <xs:complexContent>
2827         <xs:extension base="RequestAbstractType">
2828             <xs:sequence>
2829                 <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
2830             </xs:sequence>
2831         </xs:extension>
2832     </xs:complexContent>
2833 </xs:complexType>
2834
2835 <!-- Declaration of RemoveEntityResponse element -->
2836 <xs:element name="RemoveEntityResponse" type="ResponseAbstractType"/>
2837
2838 <!-- Declaration of RemoveCollectionRequest element -->
2839 <xs:element name="RemoveCollectionRequest" type="RemoveCollectionRequestType"/>
2840 <!-- Definition of RemoveCollectionRequestType -->
2841 <xs:complexType name="RemoveCollectionRequestType">
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 RemoveCollectionResponse element -->
2852 <xs:element name="RemoveCollectionResponse" type="ResponseAbstractType"/>
2853
2854 <!-- Declaration of RemoveFromCollectionRequest element -->
2855 <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
2856 <!-- Definition of RemoveFromCollectionRequestType -->
2857 <xs:complexType name="RemoveFromCollectionRequestType">
2858     <xs:complexContent>
2859         <xs:extension base="RequestAbstractType">
2860             <xs:sequence>
2861                 <xs:element ref="TargetObjectID"/>
2862                 <xs:element ref="ObjectID" maxOccurs="unbounded"/>
2863                 <xs:element ref="Subscription" minOccurs="0"/>
2864             </xs:sequence>
2865         </xs:extension>
2866     </xs:complexContent>
2867 </xs:complexType>
2868
2869 <!-- Declaration of RemoveFromCollectionResponse element -->
2870 <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>
2871
2872 <!-- Declaration of ListMembersRequest element -->
2873 <xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
2874 <!-- Definition of ListMembersRequestType -->
2875 <xs:complexType name="ListMembersRequestType">
2876     <xs:complexContent>
2877         <xs:extension base="RequestAbstractType">
2878             <xs:sequence>
2879                 <xs:element ref="TargetObjectID" minOccurs="0"/>
2880                 <xs:element ref="Subscription" minOccurs="0"/>
2881             </xs:sequence>
2882             <xs:attribute name="Structured" type="xs:anyURI" use="optional"/>
2883             <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
2884             <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
2885         </xs:extension>
2886     </xs:complexContent>
2887 </xs:complexType>
2888
2889 <!-- Declaration of ListMembersResponse element -->
2890 <xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
2891 <!-- Definition of ListMembersResponseType -->
2892 <xs:complexType name="ListMembersResponseType">

```

```

2893     <xs:complexContent>
2894         <xs:extension base="ResponseAbstractType">
2895             <xs:sequence>
2896                 <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2897             </xs:sequence>
2898         </xs:extension>
2899     </xs:complexContent>
2900 </xs:complexType>
2901
2902 <!-- Declaration of QueryObjectsRequest element -->
2903 <xs:element name="QueryObjectsRequest" type="QueryObjectsRequestType"/>
2904 <!-- Definition of QueryObjectsRequestType -->
2905 <xs:complexType name="QueryObjectsRequestType">
2906     <xs:complexContent>
2907         <xs:extension base="RequestAbstractType">
2908             <xs:sequence>
2909                 <xs:element name="Filter" type="xs:string"/>
2910                 <xs:element ref="Subscription" minOccurs="0"/>
2911             </xs:sequence>
2912             <xs:attribute name="Count" type="xs:nonNegativeInteger" use="optional"/>
2913             <xs:attribute name="Offset" type="xs:nonNegativeInteger" default="0" use="optional"/>
2914         </xs:extension>
2915     </xs:complexContent>
2916 </xs:complexType>
2917
2918 <!-- Declaration of QueryObjectsResponse element -->
2919 <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
2920 <!-- Definition of QueryObjectsResponseType -->
2921 <xs:complexType name="QueryObjectsResponseType">
2922     <xs:complexContent>
2923         <xs:extension base="ResponseAbstractType">
2924             <xs:sequence>
2925                 <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
2926             </xs:sequence>
2927         </xs:extension>
2928     </xs:complexContent>
2929 </xs:complexType>
2930
2931 <!-- Declaration of GetObjectInfoRequest element -->
2932 <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
2933 <!-- Definition of GetObjectInfoRequestType -->
2934 <xs:complexType name="GetObjectInfoRequestType">
2935     <xs:complexContent>
2936         <xs:extension base="RequestAbstractType">
2937             <xs:sequence>
2938                 <xs:element ref="TargetObjectID" minOccurs="0"/>
2939                 <xs:element ref="Subscription" minOccurs="0"/>
2940             </xs:sequence>
2941         </xs:extension>
2942     </xs:complexContent>
2943 </xs:complexType>
2944
2945 <!-- Declaration of GetObjectInfoResponse element -->
2946 <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
2947 <!-- Definition of GetObjectInfoResponseType -->
2948 <xs:complexType name="GetObjectInfoResponseType">
2949     <xs:complexContent>
2950         <xs:extension base="ResponseAbstractType">
2951             <xs:sequence>
2952                 <xs:element ref="Object" minOccurs="0"/>
2953             </xs:sequence>
2954         </xs:extension>
2955     </xs:complexContent>
2956 </xs:complexType>
2957
2958 <!-- Declaration of SetObjectInfoRequest element -->
2959 <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>

```

```

2960 <!-- Definition of SetObjectInfoRequestType -->
2961 <xs:complexType name="SetObjectInfoRequestType" >
2962   <xs:complexContent>
2963     <xs:extension base="RequestAbstractType">
2964       <xs:sequence>
2965         <xs:element ref="Object" maxOccurs="unbounded"/>
2966         <xs:element ref="Subscription" minOccurs="0"/>
2967       </xs:sequence>
2968     </xs:extension>
2969   </xs:complexContent>
2970 </xs:complexType>
2971
2972 <!-- Declaration of SetObjectInfoResponse element -->
2973 <xs:element name="SetObjectInfoResponse" type="ResponseAbstractType"/>
2974
2975 <!-- Declaration of TestMembershipRequest element -->
2976 <xs:element name="TestMembershipRequest" type="TestMembershipRequestType"/>
2977 <!-- Definition of TestMembershipRequestType -->
2978 <xs:complexType name="TestMembershipRequestType" >
2979   <xs:complexContent>
2980     <xs:extension base="RequestAbstractType">
2981       <xs:sequence>
2982         <xs:element ref="TargetObjectID" minOccurs="0"/>
2983         <xs:element ref="sec:Token"/>
2984         <xs:element ref="Subscription" minOccurs="0"/>
2985       </xs:sequence>
2986     </xs:extension>
2987   </xs:complexContent>
2988 </xs:complexType>
2989
2990 <!-- Definition of ResultType -->
2991 <xs:complexType name="ResultType">
2992   <xs:simpleContent>
2993     <xs:extension base="xs:boolean"/>
2994   </xs:simpleContent>
2995 </xs:complexType>
2996
2997 <!-- Declaration of TestMembershipResponse element -->
2998 <xs:element name="TestMembershipResponse" type="TestMembershipResponseType"/>
2999 <!-- Definition of TestMembershipResponseType -->
3000 <xs:complexType name="TestMembershipResponseType">
3001   <xs:complexContent>
3002     <xs:extension base="ResponseAbstractType">
3003       <xs:sequence>
3004         <xs:element name="Result" type="ResultType" minOccurs="0"/>
3005       </xs:sequence>
3006     </xs:extension>
3007   </xs:complexContent>
3008 </xs:complexType>
3009
3010 <!-- Declaration of ResolveIdentifierRequest element -->
3011 <xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
3012 <!-- Definition of ResolveIdentifierRequestType -->
3013 <xs:complexType name="ResolveIdentifierRequestType">
3014   <xs:complexContent>
3015     <xs:extension base="RequestAbstractType">
3016       <xs:sequence>
3017         <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
3018       </xs:sequence>
3019     </xs:extension>
3020   </xs:complexContent>
3021 </xs:complexType>
3022
3023 <!-- Declaration of ResolveInput element -->
3024 <xs:element name="ResolveInput" type="ResolveInputType"/>
3025 <!-- Definition of ResolveInputType -->
3026 <xs:complexType name="ResolveInputType">

```

```

3027     <xs:complexContent>
3028         <xs:extension base="ims:MappingInputType">
3029             <xs:sequence>
3030                 <xs:element ref="TargetObjectID" minOccurs="0"/>
3031             </xs:sequence>
3032         </xs:extension>
3033     </xs:complexContent>
3034 </xs:complexType>
3035
3036 <!-- Declaration of ResolveIdentifierResponse element -->
3037 <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType"/>
3038 <!-- Definition of ResolveIdentifierResponseType -->
3039 <xs:complexType name="ResolveIdentifierResponseType">
3040     <xs:complexContent>
3041         <xs:extension base="ResponseAbstractType">
3042             <xs:sequence>
3043                 <xs:element ref="ResolveOutput" maxOccurs="unbounded"/>
3044             </xs:sequence>
3045         </xs:extension>
3046     </xs:complexContent>
3047 </xs:complexType>
3048
3049 <!-- Declaration of ResolveOutput element -->
3050 <xs:element name="ResolveOutput" type="ims:MappingOutputType"/>
3051
3052 <!-- Declaration of Subscription element -->
3053 <xs:element name="Subscription" type="subs:SubscriptionType"/>
3054
3055 <!-- Declaration of Notification element -->
3056 <xs:element name="Notification" type="NotificationType"/>
3057 <!-- Definition of NotificationType -->
3058 <xs:complexType name="NotificationType">
3059     <xs:complexContent>
3060         <xs:extension base="subs:NotificationType">
3061             <xs:sequence>
3062                 <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded"/>
3063             </xs:sequence>
3064         </xs:extension>
3065     </xs:complexContent>
3066 </xs:complexType>
3067
3068 <!-- Declaration of ItemData element -->
3069 <xs:element name="ItemData" type="ItemDataType"/>
3070 <!-- Definition of ItemDataType -->
3071 <xs:complexType name="ItemDataType">
3072     <xs:choice>
3073         <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
3074         <xs:element ref="sec:Token" minOccurs="0"/>
3075     </xs:choice>
3076 </xs:complexType>
3077
3078 <!-- Declaration of Notify element -->
3079 <xs:element name="Notify" type="NotifyType"/>
3080 <!-- Definition of NotifyType -->
3081 <xs:complexType name="NotifyType">
3082     <xs:complexContent>
3083         <xs:extension base="RequestAbstractType">
3084             <xs:sequence>
3085                 <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
3086             </xs:sequence>
3087             <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
3088         </xs:extension>
3089     </xs:complexContent>
3090 </xs:complexType>
3091
3092 <!-- Declaration of NotifyResponse element -->
3093 <xs:element name="NotifyResponse" type="subs:NotifyResponseType"/>

```

3094
3095 </xs:schema>
3096
3097

8. Abstract WSDL

3098

3099

```
3100 <definitions
3101   name="id-wsf-ps_2006-08_wsd_interface"
3102   targetNamespace="urn:liberty:ps:2006-08"
3103   xmlns:tns="urn:liberty:ps:2006-08"
3104   xmlns="http://schemas.xmlsoap.org/wsdl/"
3105   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3106   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3107   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
3108   xmlns:ps="urn:liberty:ps:2006-08"
3109   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3110   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
3111     http://schemas.xmlsoap.org/wsdl/
3112     http://www.w3.org/2006/02/addressing/wsdl
3113     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsd.xsd">
3114
3115   <types>
3116     <xsd:schema>
3117       <xsd:import namespace="urn:liberty:ps:2006-08"
3118         schemaLocation="liberty-idwsf-people-service-v1.0.xsd"/>
3119     </xsd:schema>
3120   </types>
3121
3122   <!-- Messages -->
3123
3124   <!-- Adding a User -->
3125
3126   <message name="AddEntityRequest">
3127     <part name="body" element="ps:AddEntityRequest"/>
3128   </message>
3129
3130   <message name="AddEntityResponse">
3131     <part name="body" element="ps:AddEntityResponse"/>
3132   </message>
3133
3134   <!-- Adding a Known User -->
3135
3136   <message name="AddKnownEntityRequest">
3137     <part name="body" element="ps:AddKnownEntityRequest"/>
3138   </message>
3139
3140   <message name="AddKnownEntityResponse">
3141     <part name="body" element="ps:AddKnownEntityResponse"/>
3142   </message>
3143
3144
3145   <!-- Removing a User -->
3146
3147   <message name="RemoveEntityRequest">
3148     <part name="body" element="ps:RemoveEntityRequest"/>
3149   </message>
3150
3151   <message name="RemoveEntityResponse">
3152     <part name="body" element="ps:RemoveEntityResponse"/>
3153   </message>
3154
3155   <!-- Adding a Group -->
3156
3157   <message name="AddCollectionRequest">
3158     <part name="body" element="ps:AddCollectionRequest"/>
3159   </message>
3160
3161   <message name="AddCollectionResponse">
3162     <part name="body" element="ps:AddCollectionResponse"/>
3163   </message>
```

```
3164
3165 <!-- Removing a Group -->
3166
3167 <message name="RemoveCollectionRequest">
3168   <part name="body" element="ps:RemoveCollectionRequest"/>
3169 </message>
3170
3171 <message name="RemoveCollectionResponse">
3172   <part name="body" element="ps:RemoveCollectionResponse"/>
3173 </message>
3174
3175 <!-- Adding to a Group -->
3176
3177 <message name="AddToCollectionRequest">
3178   <part name="body" element="ps:AddToCollectionRequest"/>
3179 </message>
3180
3181 <message name="AddToCollectionResponse">
3182   <part name="body" element="ps:AddToCollectionResponse"/>
3183 </message>
3184
3185 <!-- Removing From a Group -->
3186
3187 <message name="RemoveFromCollectionRequest">
3188   <part name="body" element="ps:RemoveFromCollectionRequest"/>
3189 </message>
3190
3191 <message name="RemoveFromCollectionResponse">
3192   <part name="body" element="ps:RemoveFromCollectionResponse"/>
3193 </message>
3194
3195 <!-- Listing Members -->
3196
3197 <message name="ListMembersRequest">
3198   <part name="body" element="ps:ListMembersRequest"/>
3199 </message>
3200
3201 <message name="ListMembersResponse">
3202   <part name="body" element="ps:ListMembersResponse"/>
3203 </message>
3204
3205 <!-- Retrieving Object Info -->
3206
3207 <message name="GetObjectInfoRequest">
3208   <part name="body" element="ps:GetObjectInfoRequest"/>
3209 </message>
3210
3211 <message name="GetObjectInfoResponse">
3212   <part name="body" element="ps:GetObjectInfoResponse"/>
3213 </message>
3214
3215 <!-- Updating Object Info -->
3216
3217 <message name="SetObjectInfoRequest">
3218   <part name="body" element="ps:SetObjectInfoRequest"/>
3219 </message>
3220
3221 <message name="SetObjectInfoResponse">
3222   <part name="body" element="ps:SetObjectInfoResponse"/>
3223 </message>
3224
3225 <!-- Querying Objects -->
3226
3227 <message name="QueryObjectsRequest">
3228   <part name="body" element="ps:QueryObjectsRequest"/>
3229 </message>
3230
```

```

3231     <message name="QueryObjectsResponse">
3232         <part name="body" element="ps:QueryObjectsResponse"/>
3233     </message>
3234
3235     <!-- Testing Membership -->
3236
3237     <message name="TestMembershipRequest">
3238         <part name="body" element="ps:TestMembershipRequest"/>
3239     </message>
3240
3241     <message name="TestMembershipResponse">
3242         <part name="body" element="ps:TestMembershipResponse"/>
3243     </message>
3244
3245     <!-- Resolving Identifiers -->
3246
3247     <message name="ResolveIdentifierRequest">
3248         <part name="body" element="ps:ResolveIdentifierRequest"/>
3249     </message>
3250
3251     <message name="ResolveIdentifierResponse">
3252         <part name="body" element="ps:ResolveIdentifierResponse"/>
3253     </message>
3254
3255     <!-- Port Type -->
3256
3257     <portType name="LibertyPS1">
3258
3259         <operation name="AddEntity">
3260             <input message="tns:AddEntityRequest"
3261                 wsaw:Action="urn:liberty:ps:2006-08:AddEntityRequest"/>
3262             <output message="tns:AddEntityResponse"
3263                 wsaw:Action="urn:liberty:ps:2006-08:AddEntityResponse"/>
3264         </operation>
3265
3266         <operation name="AddKnownEntity">
3267             <input message="tns:AddKnownEntityRequest"
3268                 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityRequest"/>
3269             <output message="tns:AddKnownEntityResponse"
3270                 wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityResponse"/>
3271         </operation>
3272
3273         <operation name="RemoveEntity">
3274             <input message="tns:RemoveEntityRequest"
3275                 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityRequest"/>
3276             <output message="tns:RemoveEntityResponse"
3277                 wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityResponse"/>
3278         </operation>
3279
3280         <operation name="AddCollection">
3281             <input message="tns:AddCollectionRequest"
3282                 wsaw:Action="urn:liberty:ps:2006-08:AddCollectionRequest"/>
3283             <output message="tns:AddCollectionResponse"
3284                 wsaw:Action="urn:liberty:ps:2006-08:AddCollectionResponse"/>
3285         </operation>
3286
3287         <operation name="RemoveCollection">
3288             <input message="tns:RemoveCollectionRequest"
3289                 wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionRequest"/>
3290             <output message="tns:RemoveCollectionResponse"
3291                 wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionResponse"/>
3292         </operation>
3293
3294         <operation name="AddToCollection">
3295             <input message="tns:AddToCollectionRequest"
3296                 wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionRequest"/>
3297             <output message="tns:AddToCollectionResponse"

```

```

3298         wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionResponse"/>
3299     </operation>
3300
3301     <operation name="RemoveFromCollection">
3302         <input message="tns:RemoveFromCollectionRequest "
3303             wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionRequest"/>
3304         <output message="tns:RemoveFromCollectionResponse"
3305             wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionResponse"/>
3306     </operation>
3307
3308     <operation name="ListMembersOfCollection">
3309         <input message="tns:ListMembersRequest"
3310             wsaw:Action="urn:liberty:ps:2006-08:ListMembersRequest"/>
3311         <output message="tns:ListMembersResponse"
3312             wsaw:Action="urn:liberty:ps:2006-08:ListMembersResponse"/>
3313     </operation>
3314
3315     <operation name="GetObjectInfo">
3316         <input message="tns:GetObjectInfoRequest"
3317             wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoRequest"/>
3318         <output message="tns:GetObjectInfoResponse"
3319             wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoResponse"/>
3320     </operation>
3321
3322     <operation name="SetObjectInfo">
3323         <input message="tns:SetObjectInfoRequest"
3324             wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoRequest"/>
3325         <output message="tns:SetObjectInfoResponse"
3326             wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoResponse"/>
3327     </operation>
3328
3329     <operation name="QueryObjects">
3330         <input message="tns:QueryObjectsRequest"
3331             wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsRequest"/>
3332         <output message="tns:QueryObjectsResponse"
3333             wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsResponse"/>
3334     </operation>
3335
3336     <operation name="TestMembership">
3337         <input message="tns:TestMembershipRequest"
3338             wsaw:Action="urn:liberty:ps:2006-08:TestMembershipRequest"/>
3339         <output message="tns:TestMembershipResponse"
3340             wsaw:Action="urn:liberty:ps:2006-08:TestMembershipResponse"/>
3341     </operation>
3342
3343     <operation name="ResolveIdentifier">
3344         <input message="tns:ResolveIdentifierRequest"
3345             wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierRequest"/>
3346         <output message="tns:ResolveIdentifierResponse"
3347             wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierResponse"/>
3348     </operation>
3349
3350 </portType>
3351
3352 <!--
3353 An example of a binding and service that can be used with this
3354 abstract service description is provided below.
3355 -->
3356
3357 <binding name="PeopleServiceSoapBinding" type="tns:LibertyPS1">
3358
3359     <soap:binding style="document"
3360         transport="http://schemas.xmlsoap.org/soap/http"/>
3361
3362     <operation name="AddEntity">
3363         <soap:operation/>
3364         <input> <soap:body use="literal"/> </input>

```

```
3365         <output> <soap:body use="literal"/> </output>
3366     </operation>
3367
3368     <operation name="AddKnownEntity">
3369         <input> <soap:body use="literal"/> </input>
3370         <output> <soap:body use="literal"/> </output>
3371     </operation>
3372     <operation name="RemoveEntity">
3373         <input> <soap:body use="literal"/> </input>
3374         <output> <soap:body use="literal"/> </output>
3375     </operation>
3376     <operation name="AddCollection">
3377         <input> <soap:body use="literal"/> </input>
3378         <output> <soap:body use="literal"/> </output>
3379     </operation>
3380     <operation name="RemoveCollection">
3381         <input> <soap:body use="literal"/> </input>
3382         <output> <soap:body use="literal"/> </output>
3383     </operation>
3384     <operation name="AddToCollection">
3385         <input> <soap:body use="literal"/> </input>
3386         <output> <soap:body use="literal"/> </output>
3387     </operation>
3388     <operation name="RemoveFromCollection">
3389         <input> <soap:body use="literal"/> </input>
3390         <output> <soap:body use="literal"/> </output>
3391     </operation>
3392     <operation name="ListMembersOfCollection">
3393         <input> <soap:body use="literal"/> </input>
3394         <output> <soap:body use="literal"/> </output>
3395     </operation>
3396     <operation name="GetObjectInfo">
3397         <input> <soap:body use="literal"/> </input>
3398         <output> <soap:body use="literal"/> </output>
3399     </operation>
3400     <operation name="SetObjectInfo">
3401         <input> <soap:body use="literal"/> </input>
3402         <output> <soap:body use="literal"/> </output>
3403     </operation>
3404     <operation name="QueryObjects">
3405         <input> <soap:body use="literal"/> </input>
3406         <output> <soap:body use="literal"/> </output>
3407     </operation>
3408     <operation name="TestMembership">
3409         <input> <soap:body use="literal"/> </input>
3410         <output> <soap:body use="literal"/> </output>
3411     </operation>
3412     <operation name="ResolveIdentifier">
3413         <input> <soap:body use="literal"/> </input>
3414         <output> <soap:body use="literal"/> </output>
3415     </operation>
3416
3417 </binding>
3418
3419 <service name="PeopleService">
3420     <port name="PeoplePort" binding="ps:PeopleServiceSoapBinding">
3421
3422         <!-- Modify with the REAL SOAP endpoint -->
3423
3424         <soap:address location="http://example.com/peopleservice"/>
3425     </port>
3426 </service>
3427
3428 </definitions>
3429
3430
```

References

Normative

3431

3432

3433 [LibertyAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,
3434 Single Sign-On, and Identity Mapping Services Specification," Version 2.0-errata-v1.0, Liberty Alliance
3435 Project (28 November, 2006). <http://www.projectliberty.org/specs>

3436 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-
3437 errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>

3438 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
3439 2006). <http://www.projectliberty.org/specs>

3440 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
3441 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>

3442 [LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0 errata v1.0, Liberty Alliance Project (21 April, 2007).
3443 <http://www.projectliberty.org/specs>

3444 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0,
3445 Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>

3446 [LibertySUBS] Kellomäki, Sampo, eds. "Liberty ID-WSF Subscriptions and Notifications," Version 1.0, Liberty
3447 Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>

3448 [LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata,"
3449 Version 1.0, Liberty Alliance Project (13 April, 2007). <http://www.projectliberty.org/specs>

3450 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
3451 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>

3452 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
3453 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
3454 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-
3455 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)

3456 [SAMLBind2] Cantor, Scott, Hirsch, Frederick, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March
3457 2005). "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
3458 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf](http://docs.oasis-
3459 open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf)

3460 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
3461 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
3462 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards
3463 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>

3464 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.
3465 World Wide Web Consortium W3C Recommendation (9 May 2006). [http://www.w3.org/TR/2006/REC-ws-
addr-core-20060509/](http://www.w3.org/TR/2006/REC-ws-
3466 addr-core-20060509/)

3467 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
3468 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
3469 <http://www.w3.org/TR/2004/REC-xml-20040204>

3470 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
3471 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
3472 <http://www.w3.org/TR/xmlschema-1/>