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

Contents

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

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

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

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

1.3. Terminology

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

1.4. Namespaces

The following namespaces are used in the schema definitions:

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

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

```
<xs:complexType name="TagType">
  <xs:attribute name="Ref" type="xs:anyURI" use="required"/>
</xs:complexType>
```

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

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

2.1.7. `<ObjectRef>` Element

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

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

2.2. `<Token>` Element

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

The `<sec:Token>` element is used by the PS provider to return requested identity tokens to the WSC, either in a `<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*

- *urn:liberty:ps:2006-08:GetObjectInfoResponse*
- *urn:liberty:ps:2006-08:SetObjectInfoRequest*
- *urn:liberty:ps:2006-08:SetObjectInfoResponse*
- *urn:liberty:ps:2006-08:QueryObjectsRequest*
- *urn:liberty:ps:2006-08:QueryObjectsResponse*
- *urn:liberty:ps:2006-08:TestMembershipRequest*
- *urn:liberty:ps:2006-08:TestMembershipResponse*
- *urn:liberty:ps:2006-08:ResolveIdentiferRequest*
- *urn:liberty:ps:2006-08:ResolveIdentifierResponse*
- *urn:liberty:ps:2006-08:Notify*
- *urn:liberty:ps:2006-08:NotifyResponse*

3.4. Request and Response Abstract Types

3.4.1. Complex Type RequestAbstractType

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

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

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

```

<!-- Definition of RequestAbstractType -->
<xs:complexType name="RequestAbstractType" abstract="true">
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

```

3.4.2. Complex Type ResponseAbstractType

All PS response messages are of types that are derived from the abstract **`ResponseAbstractType`** complex type. This type defines common attributes and elements that are associated with all PS responses:

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

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

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

```

<!-- Definition of ResponseAbstractType -->
<xs:complexType name="ResponseAbstractType" abstract="true">
   <xs:sequence>
      <xs:element ref="lu:Status"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

```

3.5. Status

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

- CannotFindIDP
- CannotFindObject
- CannotResolveToken
- CircularCollection
- DuplicateObject
- Failed
- InvalidNodeType
- InvalidObjectID
- ObjectIsCollection
- ObjectIsEntity
- OK
- OKButNoSubscription
- NoResults
- NoSubscribeWithOffset
- NoTargetSpecified
- PartialSuccess
- PolicyDoesNotAllow
- ResolveIdentifierNotSupported
- SubscribeToChildrenOnly

- Timeout
- UnexpectedError
- UnrecognizedFilter
- UnrecognizedNamespace
- UnspecifiedError

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

OK	The value *OK* means that the processing of the request message has succeeded. A second level status code MAY be used to indicate some special cases, but the processing of the request message has succeeded.
OKButNoSubscription	The value *OKButNoSubscription* means that the processing of the primary request message has succeeded but a Subscription request within that message has been rejected. A second level status code MAY be used to indicate some special cases.
PartialSuccess	The value *PartialSuccess* means that the processing of the request message has partially succeeded. A second level status code MAY be used to indicate which processes failed to be processed.
Failed	The value *Failed* means that the processing of the request message has failed. A second level status code MAY be used to indicate the reason for the failure.

3.6. Identity Token Policy

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

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

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

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

3.7. Success & Failure

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

3.8. Subscription and Notification

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

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

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

3.8.1. `<Subscription>` Element

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

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

```

 <xs:element name="Subscription" type="subs:SubscriptionType"/>

```

3.8.1.1. Selecting Objects

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

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

3.8.1.2. Triggers

This specification defines no triggers.

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

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

3.8.1.3. Subscription Start

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

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

3.8.1.4. Subscription Aggregation

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

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

3.8.1.5. Subscription Expiration

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

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

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

3.8.1.6. Subscription Querying and Management

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

3.8.1.7. Including Data

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

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

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

3.8.2. Notify and NotifyResponse Messages

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

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

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

```

<xs:element name="Notify" type="NotifyType"/>

<xs:complexType name="NotifyType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="NotifyResponse" type="subs:NotifyResponseType"/>

```

3.8.3. `<Notification>` Element

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

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

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

```

<xs:element name="Notification" type="NotificationType"/>

<xs:complexType name="NotificationType">
  <xs:complexContent>
    <xs:extension base="subs:NotificationType">
      <xs:sequence>
        <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

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

```

<xs:element name="ItemData" type="ItemDataType"/>

<xs:complexType name="ItemDataType">
  <xs:choice>
    <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="sec:Token" minOccurs="0"/>
  </xs:choice>
</xs:complexType>

```

3.9. Adding an Entity

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

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

3.9.1. wsa:Action Values

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

3.9.2. AddEntityRequest Message

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

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

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

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

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

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

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

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

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

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

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

```
<!-- Declaration of AddEntityRequest element -->
<xs:element name="AddEntityRequest" type="AddEntityRequestType"/>
<!-- Definition of AddEntityRequestType -->
<xs:complexType name="AddEntityRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Object"/>
            <xs:element ref="PStoSPRedirectURL" minOccurs="0"/>
            <xs:element ref="CreatePSObject" minOccurs="0"/>
            <xs:element ref="Subscription" minOccurs="0"/>
            <xs:element ref="TokenPolicy" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<AddEntityRequest>
   <Object NodeType="urn:liberty:ps:entity">
      <DisplayName>Alison</DisplayName>
   </Object>
   <PStoSPRedirectURL>some SP URL</PStoSPRedirectURL>
</AddEntityRequest>
```

3.9.3. AddEntityResponse Message

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

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

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

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

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

This element is defined to offer better protection against identity theft attacks during the invitation process. See Section 4.1 for more detail.

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

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

```
<!-- Declaration of AddEntityResponse element -->
<xs:element name="AddEntityResponse" type="AddEntityResponseType"/>
<!-- Definition of AddEntityResponseType -->
<xs:complexType name="AddEntityResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element ref="Object" minOccurs="0"/>
            <xs:element ref="SPtoPSRedirectURL" minOccurs="0"/>
            <xs:element ref="QueryString" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<AddEntityResponse>
   <Status code="OK"/>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/kudfhgs</ObjectID>
      <DisplayName>Alison</DisplayName>
   </Object>
   <SPtoPSRedirectURL>some PS URL</SPtoPSRedirectURL>
</AddEntityResponse>
```

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.

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

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

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

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

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

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

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

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

3.10. Adding a Known Entity

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

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

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

3.10.1. wsa:Action Values

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

3.10.2. AddKnownEntityRequest Message

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

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

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

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

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

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

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

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

`<sec:TokenPolicy>` **[Optional]** The `<sec:TokenPolicy>` element is used as a container for a WSC's requirements to an identity token.

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

```

<!-- Declaration of AddKnownEntityRequest element -->
<xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType"/>
<!-- Definition of AddKnownEntityRequestType -->
<xs:complexType name="AddKnownEntityRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Object"/>
            <xs:element ref="sec:Token"/>
            <xs:element ref="CreatePSObject" minOccurs="0"/>
            <xs:element ref="Subscription" minOccurs="0"/>
            <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

```

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

```

<AddKnownEntityRequest>
   <Object NodeType="urn:liberty:ps:entity">
      <DisplayName>Bob</DisplayName>
   </Object>
   <sec:Token>
 <saml:Assertion>
   <saml:Subject>
        <saml:NameID></saml:NameID>
      </saml:Subject>
 </saml:Assertion>
```

```
   </sec:Token>
</AddKnownEntityRequest>
```

3.10.3. AddKnownEntityResponse Message

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

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

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

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

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

This element is defined to offer better protection against identity theft attacks during the invitation process. See Section 4.1 for more detail.

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

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

```
<!-- Declaration of AddKnownEntityResponse element -->
<xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType"/>
<!-- Definition of AddKnownEntityResponseType -->
<xs:complexType name="AddKnownEntityResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element ref="Object" minOccurs="0"/>
            <xs:element ref="SPtoPSRedirectURL" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

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

```
      <ObjectID>https://ps.com/lafnervf</ObjectID>
      <DisplayName>Bob</DisplayName>
   </Object>
</AddKnownEntityResponse>
```

3.10.4. Processing Rules

The WSC:

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

  When the token is not an identity token (as is the likely case when the known identifier is provided by the inviting user), the WSC SHOULD use a SAML `<saml:NameID>` element within the `<Token>` element. If the WSC knows the format of the known identifier, it SHOULD use the appropriate value for the Format attribute on the `<saml:NameID>` element.
- MUST, if a SAML `<Assertion>` is used as the identity token format, specify the known identifier in that assertion's `<Subject>` element.
- MUST include at least one `<DisplayName>` for the invited user within the `<Object>` element. This element contains the friendly name that the user desires be used for the created user `Object`.
- MAY, if it desires that the PS provider create (or verify the existence of) 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>` message - this 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.

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

- MAY include either or both of a `<SPtoPSRedirectURL>` or `<QueryString>` element in the `<AddKnownEntityResponse>` message returned to the calling SP.
- SHOULD, if the `<AddKnownEntityRequest>` message contained a `<Subscription>` element, send a `<ims:IdentityMappingRequest>` message to that IDP requesting an identity token for the user for which the federated identifier was established but in the namespace of the requesting SP.

  This `<ims:IdentityMappingRequest>` message to the IDP MUST include any policy directives present in the `<AddKnownEntityRequest>`.
- SHOULD, if the `<AddKnownEntityRequest>` message contained a `<Subscription>` element and the `<ims:IdentityMappingRequest>` message to the IDP resulted in an identity token for the user being returned, forward on this identity token to the SP within a `<Notification>` message corresponding to the original `<Subscription>` element.

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

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

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

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

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

3.11. Removing an Entity

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

3.11.1. wsa:Action Values

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

3.11.2. RemoveEntityRequest Message

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

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

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

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

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

```
<!-- Declaration of RemoveEntityRequest element -->
<xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType"/>
<!-- Definition of RemoveEntityRequestType -->
<xs:complexType name="RemoveEntityRequestType">
   <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 `<RemoveEntityRequest>` message used to remove an `Object` for a user.

```
<RemoveEntityRequest>
   <TargetObjectID>https://ps.com/lafnervf</TargetObjectID>
</RemoveEntityRequest>
```

3.11.3. RemoveEntityResponse Message

A PS provider responds to a `<RemoveEntityRequest>` message with a `<RemoveEntityResponse>` element. A PS provider removes the specified `<Object>` and responds with a status of this process based on the processing rules described in section Section 3.11.4.

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

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

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

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

```
<RemoveEntityResponse>
   <Status code="OK"/>
</RemoveEntityResponse>
```

3.11.4. Processing Rules

The WSC:

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

The PS provider:

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

    - NotDirectChild

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

3.12. Adding a Collection

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

3.12.1. wsa:Action Values

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

3.12.2. AddCollectionRequest Message

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

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

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

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

`<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 group object being added changes.

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

```
<!-- Declaration of AddCollectionRequest element -->
<xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
<!-- Definition of AddCollectionRequestType -->
<xs:complexType name="AddCollectionRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Object"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<AddCollectionRequest>
   <Object NodeType="urn:liberty:ps:collection">
      <DisplayName>Soccer Team</DisplayName>
   </Object>
</AddCollectionRequest>
```

3.12.3. AddCollectionResponse Message

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

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

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

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

```
<!-- Declaration of AddCollectionResponse element -->
<xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>
<!-- Definition of AddCollectionResponseType -->
<xs:complexType name="AddCollectionResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element ref="Object" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<AddCollectionResponse>
   <Status code="OK"/>
   <Object NodeType="urn:liberty:ps:collection">
      <ObjectID>https://ps.com/roqlsfof</ObjectID>
      <DisplayName>Soccer Team</DisplayName>
   </Object>
</AddCollectionResponse>
```

3.12.4. Processing Rules

The WSC:

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

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

- MUST return an `<Object>` element within the `<AddCollectionResponse>` message with the same `<DisplayName>` as specified on the `<AddCollectionRequest>`.
- 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

A PS provider responds to a `<RemoveCollectionRequest>` message with a `<RemoveCollectionResponse>` element. A PS provider removes the specified `<Object>` and responds a status of this process based on the processing rules described in Section 3.11.4.

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

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

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

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

```
<RemoveCollectionResponse>
   <Status code="OK"/>
</RemoveCollectionResponse>
```

3.13.4. Processing Rules

The WSC:

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

The PS provider:

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

3.14. Adding to a Collection

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

3.14.1. wsa:Action Values

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

3.14.2. AddToCollectionRequest Message

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

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

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

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

`<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 membership of the targeted group changes.

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

```

<!-- Declaration of AddToCollectionRequest element -->
<xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>
<!-- Definition of AddToCollectionRequestType -->
<xs:complexType name="AddToCollectionRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="TargetObjectID"/>
            <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
       </xs:extension>
   </xs:complexContent>
</xs:complexType>

```

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

```

<AddToCollectionRequest>
   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectID>
   <ObjectID>https://ps.com/qaf3eflo</ObjectID>
   <ObjectID>https://ps.com/bzpfnrns</ObjectID>
   <ObjectID>https://ps.com/nsdflfss</ObjectID>
</AddToCollectionRequest>

```

3.14.3. AddToCollectionResponse Message

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

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

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

```

<!-- Declaration of AddToCollectionResponse element -->
<xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>

```

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

```

<AddToCollectionResponse>
```

```
   <Status code="OK"/>
</AddToCollectionResponse>
```

3.14.4. Processing Rules

The WSC:

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

The PS provider:

- MUST, if the `Object` specified by the value of the `<TargetObjectID>` element has *urn:liberty:ps:entity* as the value of its `NodeType` attribute, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - ObjectIsEntity
- MUST, if the `Object` specified by the value of an `<ObjectID>` element is already a member of the targeted collection, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - DuplicateObject
- MUST NOT allow the creation of circular collections. A circular collection is one which includes, at any layer of the structure below it, a reference to the same collection such that a dereference of the collection would result in an infinite loop. If the PS provider receives an `<AddToCollectionRequest>` message that would result in a circular collection, it MUST respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - CircularCollection

3.15. Removing from a Collection

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

3.15.1. wsa:Action Values

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

3.15.2. RemoveFromCollectionRequest Message

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

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

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

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

`<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 membership of the targeted group changes.

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

```

<!-- Declaration of RemoveFromCollectionRequest element -->
<xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
<!-- Definition of RemoveFromCollectionRequestType -->
<xs:complexType name="RemoveFromCollectionRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="TargetObjectID"/>
            <xs:element ref="ObjectID" maxOccurs="unbounded"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
       </xs:extension>
   </xs:complexContent>
</xs:complexType>

```

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

```

<RemoveFromCollectionRequest>
   <TargetObjectID>https://ps.com/roqlsfof</TargetObjectI D>
   <ObjectID>https://ps.com/qaf3eflo</ObjectID>
   <ObjectID>https://ps.com/bzpfnrn s</ObjectID>
   <ObjectID>https://ps.com/nsdflfss</ObjectI D>
</RemoveFromCollectionRequest>

```

3.15.3. RemoveFromCollectionResponse Message

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

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

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

```

<!-- Declaration of RemoveFromCollectionResponse element -->
<xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>

```

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

```

<RemoveFromCollectionResponse>
```

```
   <Status code="OK"/>
</RemoveFromCollectionResponse>

```

3.15.4. Processing Rules

The WSC:

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

The PS provider:

- MUST, if the `Object` specified by the value of the `<TargetObjectID>` element has *urn:liberty:ps:entity* as the value of its `NodeType` attribute, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - ObjectIsEntity
- MUST, if the `Object` specified by the value of an `<ObjectID>` element is not a member of the targeted collection, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - CannotFindObject

3.16. Listing Members

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

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

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

3.16.1. wsa:Action Values

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

3.16.2. ListMembersRequest Message

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

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

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

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

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

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

`<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 membership of the targeted group changes.

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

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

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

```

<!-- Declaration of ListMembersRequest element -->
<xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
<!-- Definition of ListMembersRequestType -->
<xs:complexType name="ListMembersRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="TargetObjectID" minOccurs="0"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="Structured" type="xs:string" use="optional"/>
         <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>

```

3.16.2.1. Structured Attribute

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

The `Structured` attribute takes three possible values

children	The WSC is indicating that it desires only the direct child collection and entity objects of the targeted object. The default value for the `Structured` attribute is *children*.
tree	the WSC is indicating that it desires the full tree structure of the targetted object.

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

3.16.2.2. Count and Offset Attributes

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

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

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

3.16.3. ListMembersResponse Message

A PS provider responds to a `<ListMembersRequest>` message with a `<ListMembersResponse>` message containing the appropriate set of `<Object>` elements.

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

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

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

```
<!-- Declaration of ListMembersResponse element -->
<xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
<!-- Definition of ListMembersResponseType -->
<xs:complexType name="ListMembersResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

3.16.4. Examples

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

```
<Object NodeType="urn:liberty:ps:entity">
  <ObjectID>https://ps.com/lsdjfojd</Object ID>
  <DisplayName>Mary</DisplayName>
</Object>
<Object NodeType="urn:liberty:ps:entity ">
  <ObjectID>https://ps.com/sijfsfsf</ObjectID>
  <DisplayName>Bob</DisplayName>
</Object>
<Object NodeType="urn:liberty:ps:entity">
  <ObjectID>https://ps.com/reremvls</ObjectID>
  <DisplayName>Nick</DisplayName>
</Object>
```

```
<Object NodeType="urn:liberty:ps:entity">
 <ObjectID>https://ps.com/soijfsfd</ObjectID>
 <DisplayName>JoJo</DisplayName>
</Object>
<Object NodeType="urn:liberty:ps:entity">
 <ObjectID>https://ps.com/sdosafms</ObjectID>
 <DisplayName>Taro</DisplayName>
</Object>
<Object NodeType="urn:liberty:ps:entity">
 <ObjectID>https://ps.com/lgsdfsfd</ObjectID>
 <DisplayName>Hanako</DisplayName>
</Object>
<Object NodeType="urn:liberty:ps:collection">
 <ObjectID>https://ps.com/eiruvoie</ObjectID>
 <DisplayName>Soccer Team</DisplayName>
 <Object NodeType="urn:liberty:ps:collection">
  <ObjectID>https://ps.com/nmerflas</ObjectID>
  <DisplayName>Starting Members</DisplayName>
  <ObjectRef Ref="https://ps.com/lsdjfojd"/>
  <ObjectRef Ref="https://ps.com/sijfsfsf"/>
 </Object>
  <ObjectRef Ref="https://ps.com/reremvls"/>
  <ObjectRef Ref="https://ps.com/soijfsfd"/>
</Object>
<Object NodeType="urn:liberty:ps:collection">
 <ObjectID>https://ps.com/zxlidfdf</ObjectID>
 <DisplayName>Family</DisplayName>
 <ObjectRef Ref="https://ps.com/sdosafms"/>
 <ObjectRef Ref="https://ps.com/lgsdfsfd"/>
</Object>
```

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

```
<ListMembersRequest Structured="entities">
  <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
</ListMembersRequest>
```

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

```
<ListMembersResponse>
  <Status code="OK"/>
  <Object NodeType="urn:liberty:ps:entity">
    <ObjectID>https://ps.com/lsdjfojd</ObjectID>
    <DisplayName>Mary</DisplayName>
  </Object>
  <Object NodeType="urn:liberty:ps:entity">
    <ObjectID>https://ps.com/sijfsfsf</ObjectID>
    <DisplayName>Bob</DisplayName>
  </Object>
  <Object NodeType="urn:liberty:ps:entity">
    <ObjectID>https://ps.com/reremvls</ObjectID>
    <DisplayName>Nick</DisplayName>
  </Object>
  <Object NodeType="urn:liberty:ps:entity">
    <ObjectID>https://ps.com/soijfsfd</ObjectID>
    <DisplayName>JoJo</DisplayName>
  </Object>
</ListMembersResponse>
```

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

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

```
<ListMembersResponse>
   <Status code="OK"/>
   <Object NodeType="urn:liberty:ps:collection">
      <ObjectID>https://ps.com/nmerflas</ObjectID>
      <DisplayName>Starting Members</DisplayName>
         <Object NodeType="urn:liberty:ps:entity">
            <ObjectID>https://ps.com/lsdjfojd</ObjectID>
            <DisplayName>Mary</DisplayName>
         </Object>
         <Object NodeType="urn:liberty:ps:entity">
            <ObjectID>https://ps.com/sijfsfsf</ObjectID>
            <DisplayName>Bob</DisplayName>
         </Object>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/reremvls</ObjectID>
      <DisplayName>Nick</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/soijfsfd</ObjectID>
      <DisplayName>JoJo</DisplayName>
   </Object>
</ListMembersResponse>
```

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

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

```
<ListMembersResponse>
   <Status code="OK"/>
   <Object NodeType="urn:liberty:ps:collection">
      <ObjectID>https://ps.com/nmerflas</ObjectID>
      <DisplayName>Starting Members</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/reremvls</ObjectID>
      <DisplayName>Nick</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/soijfsfd</ObjectID>
      <DisplayName>JoJo</DisplayName>
   </Object>
</ListMembersResponse>
```

3.16.5. Processing Rules

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

- The PS provider MUST, if the `<ListMembersRequest>` contains a Subscription and the Offset attribute has any value other than "0," and it is otherwise capable of returning results, return those results as indicated by the Count and Offset attributes, but still reject the Subscription by responding with "OKButNoSubscription" as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
    - NoSubscribeWithOffset
- The PS provider SHOULD dereference all `<ObjectRef>` elements and replace them with the appropriate `<Object>` elements before returning.
- At any one level of the tree, the PS provider SHOULD remove all duplicate `<Object>` elements before returning.
- If the `Structured` attribute is set as *tree*, the PS provider MUST return all the direct child and descendant `<Object>` elements of the specified `Object`.
- If the `Structured` attribute is not set in the request, the PS provider MUST process it as if it is set as *children*.
- If the `Structured` attribute is set as *entities*, the PS provider MUST return all the direct child and descendant entity `<Object>` elements of the specified `Object` (subject to the restriction defined by the `Count` attribute, if present). Any collection `<Object>` elements MUST be removed and only entity `<Object>` elements returned.
- If the `Structured` attribute is set as *children*, the PS provider MUST return all the direct child collection and entity `<Object>` elements of the specified `Object`.
- If the Count attribute is included in a `<ListMembersRequest>` message, the PS provider SHOULD NOT respond with more objects than specified. A PS provider MAY return a smaller number of objects than specified by the Count attribute.
- If the Offset attribute is included in a `<ListMembersRequest>` message, the PS provider SHOULD respond with a list of `<Object>` elements, the first in the list being that `<Object>` element whose position in the complete list is specified by the value of the Offset attribute.
- If a PS provider receives a `<ListMembersRequest>` message on which the value of `<TargetObjectID>` matches that of an `<ObjectID>` element of a given `Object`, and if the value of the `NodeType` attribute of the matched `Object` is *urn:liberty:ps:entity*, then the PS provider MUST respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
    - ObjectIsEntity
- The PS Provider MUST, if unable to find the targeted `Object`, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
    - CannotFindObject
- When the targeted `Object` has no child Objects, the PS provider MUST respond with *OK* as the code attribute of the top level `<lu:Status>` element with no `<Object>` element in the response.
- The PS provider SHOULD, if the `<ListMembersRequest>` contains a Subscription and the Structured attribute has any value other than "children," and it is otherwise capable of returning results, return those results but still reject the Subscription by responding with "OKButNoSubscription" as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
    - SubscribeToChildrenOnly

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

3.17. Retrieving Info

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

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

3.17.1. wsa:Action Values

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

3.17.2. GetObjectInfoRequest Message

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

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

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

`<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 information (but not membership) of the targeted `Object` changes.

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

```

<!-- Declaration of GetObjectInfoRequest element -->
<xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
<!-- Definition of GetObjectInfoRequestType -->
<xs:complexType name="GetObjectInfoRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="TargetObjectID"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

```

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

```

<GetObjectInfoRequest>
   <TargetObjectID>https://ps.com/eiruvoie</TargetObjectID>
</GetObjectInfoRequest>

```

3.17.3. GetObjectInfoResponse Message

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

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

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

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

```
<!-- Declaration of GetObjectInfoResponse element -->
<xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
<!-- Definition of GetObjectInfoResponseType -->
<xs:complexType name="GetObjectInfoResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element ref="Object" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<GetObjectInfoResponse>
   <Status code="OK"/>
   <Object NodeType="urn:liberty:ps:collection">
      <ObjectID>https://ps.com/eiruvoie</ObjectID>
      <DisplayName>Soccer Team</DisplayName>
   </Object>
</GetObjectInfoResponse>
```

3.17.4. Processing Rules

A PS provider:

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

3.18. Updating Info

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

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

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

3.18.1. wsa:Action Values

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

3.18.2. SetObjectInfoRequest Message

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

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

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

`<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 information of the targeted `Object` changes.

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

```

<!-- Declaration of SetObjectInfoRequest element -->
<xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
<!-- Definition of SetObjectInfoRequestType -->
<xs:complexType name="SetObjectInfoRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Object" maxOccurs="unbounded"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

```

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

```

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

```
      <ObjectID>https://ps.com/eiruvoie</ObjectID>
      <DisplayName>Baseball Team</DisplayName>
   </Object>
</SetObjectInfoRequest>
```

3.18.3. SetObjectInfoResponse Message

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

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

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

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

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

```
<SetObjectInfoResponse>
   <Status code="OK"/>
</SetObjectInfoResponse>
```

3.18.4. Processing Rules

A PS provider:

- MUST, if it can not find the target `Object` specified with the `ObjectID`, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - CannotFindObject
- MUST, if it finds that the value of the specified `NodeType` attribute is different from the value of the `NodeType` attribute of the target `Object` specified with the `ObjectID`, respond with *Failed* as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:
  - InvalidNodeType
- MUST, if it finds that a WSC specifies a child `<Object>` element, `<ObjectRef>` elements, `CreatedDateTime` 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

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

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

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

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

```

<QueryObjectsRequest>
   <Filter>//ps:Object[@NodeType='urn:liberty:ps:entity']</Filter>
</QueryObjectsRequest>

```

3.19.2.2. Count and Offset Attributes

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

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

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

3.19.3. QueryObjectsResponse Message

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

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

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

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

```
<!-- Declaration of QueryObjectsResponse element -->
<xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
<!-- Definition of QueryObjectsResponseType -->
<xs:complexType name="QueryObjectsResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<QueryObjectsResponse>
   <Status code="OK"/>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/lsdjfojd</ObjectID>
      <DisplayName>Mary</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/sijfsfsf</ObjectID>
      <DisplayName>Bob</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/reremvls</ObjectID>
      <DisplayName>Nick</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/soijfsfd</ObjectID>
      <DisplayName>JoJo</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/sdosafms</ObjectID>
      <DisplayName>Taro</DisplayName>
   </Object>
   <Object NodeType="urn:liberty:ps:entity">
      <ObjectID>https://ps.com/lgsdfsfd</ObjectID>
      <DisplayName>Hanako</DisplayName>
   </Object>
</QueryObjectsResponse>
```

3.19.4. Processing Rules

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

  - NoResults

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

  The PS provider MAY, if the `<QueryObjectsRequest>` contained a Subscription it does not wish to accept and it is otherwise capable of returning results, return those results but still reject the Subscription by responding with "OKButNoSubscription" as the code attribute of the top level `<lu:Status>` element. A second level `<lu:Status>` MAY be inserted. If so, the code attribute of that second level `<lu:Status>` element MUST be set with the following status code:

  - PolicyDoesNotAllow

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

3.20. Testing Membership

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

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

3.20.1. wsa:Action Values

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

3.20.2. TestMembershipRequest Message

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

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

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

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

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

`<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 results of the test changes.

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

```
<!-- Declaration of TestMembershipRequest element -->
<xs:element name="TestMembershipRequest" type="TestMembershipRequestType"/>
<!-- Definition of TestMembershipRequestType -->
<xs:complexType name="TestMembershipRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="TargetObjectID" minOccurs="0"/>
            <xs:element ref="Token"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<TestMembershipRequest>
   <TargetObjectID>https://ps.com/eiruvoie</TargetObjec tID>
   <Token>

   </Token>
</TestMembershipRequest>
```

3.20.3. TestMembershipResponse Message

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

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

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

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

```
<!-- Definition of ResultType -->
<xs:complexType name="ResultType">
   <xs:complexContent>
      <xs:extension base="xs:boolean"/>
   </xs:complexContent>
</xs:complexType>

<!-- Declaration of TestMembershipResponse element -->
<xs:element name="TestMembershipResponse" type="TestResponseType"/>
<!-- Definition of TestMembershipResponseType -->
<xs:complexType name="TestMembershipResponseType ">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element name="Result" type="ResultType" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

```
<TestMembershipResponse>
   <Status code="OK"/>
   <Result>true</Result>
</TestMembershipResponse>
```

3.20.4. Processing Rules

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

- ObjectIsEntity

3.21. Resolving Objects

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

3.21.1. wsa:Action Values

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

3.21.2. ResolveIdentifierRequest Message

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

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

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

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

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

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

```
<!-- Declaration of ResolveIdentifierRequest element -->
<xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
<!-- Definition of ResolveIdentifierRequestType -->
<xs:complexType name="ResolveIdentifierRequest Type">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

3.21.2.1. ResolveInput element

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

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

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

```
<!-- Declaration of ResolveInput element -->
<xs:element name="ResolveInput" type="ResolveInputType"/>
<!-- Definition of ResolveInputType -->
<xs:complexType name="ResolveInputType">
   <xs:complexContent>
      <xs:extension base="ims:MappingInputType">
         <xs:sequence>
            <xs:element ref="TargetObjectID"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

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

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

```
<ResolveIdentifierRequest>
   <ResolveInput>
     <TargetObjectID>https://ps.com/lgsdfsfd</TargetObjectID>
     <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
       <samlp2:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
            SPNameQualifier="https://psa.com"/>
     </sec:TokenPolicy>
   </ResolveInput>
</ResolveIdentifierRequest>
```

3.21.3. ResolveIdentifierResponse Message

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

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

```

   <!-- Declaration of ResolveIdentifierResponse element -->
   <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType"/>
   <!-- Definition of ResolveIdentifierResponseType -->
   <xs:complexType name="ResolveIdentifierResponseType">
      <xs:complexContent>
         <xs:extension base="ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="ResolveOutput" maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

```

3.21.3.1. ResolveOutput element

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

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

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

```

   <!-- Declaration of ResolveOutput element -->
   <xs:element name="ResolveOutput" type="ims:MappingOutputType"/>

```

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

```

<ResolveIdentifierResponse>
   <Status code="OK"/>
   <ResolveOutput>
     <Token>

     </Token>
   </ResolveOutput>
</ResolveIdentifierResponse>
```

3.21.4. Processing Rules

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

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

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

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

  - CannotResolveToken

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

  - CannotResolveToken

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

  - 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).

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

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

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

4.3. Consent Model

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

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

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

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

4.4. Elements Supporting Invitation

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

4.4.1. PStoSPRedirectURL element

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

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

4.4.2. `<SPtoPSRedirectURL>` element

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

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

The PS provider 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 this url for a reasonable length of time.

4.4.2.1. Processing Rules

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

- MUST, after appropriately informing and "consenting" the invited user, direct the user agent to the address previously specified within the `<SPtoPSRedirectURL>` element.

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

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

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

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

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

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

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

4.4.3. `<QueryString>` element

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

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

4.4.3.1. Schema

```
<xs:element name="QueryString" type="QueryStringType"/>
   <xs:complexType name="QueryStringType">
      <xs:simpleContent>
         <xs:extension base="xs:string"/>
      </xs:simpleContent>
```

```
  </xs:complexType>

```

4.4.3.2. Formatting Rules

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

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

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

```

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

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

4.4.3.3. Processing Rules

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

- MUST create an artifact string in accordance with the SAML Artifact Binding (see [SAMLBind2]).
- MUST insert this string within an `<QueryString>` element in the `<AddEntityResponse>` or `<AddKnownEntityResponse>` message.
- MUST be prepared for, at some point in the future, an IDP to send an `<samlp:ArtifactResolve>` message as a consequence of the invited user presenting the contents of any specified `<QueryString>` element to the IDP. As it may be some time before the invited user does present the artifact to their IDP, the PS provider SHOULD store the artifact for a reasonable length of time.

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

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

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

- MUST extract the contents of the `<QueryString>` element
- MUST attempt to commmunicate the extracted string to the invited user.

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

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

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

  If the value of the presented query string included any relay state information, the binding by which the `<samlp:Response>` message is delivered to the PS MUST support the communication of this relay state information back to the PS provider.

Once the invited user has been redirected to the PS provider and the PS provider has obtained the `<samlp:Response>`, the PS provider:

- MUST extract the name identifier from within the `<Subject>` of the `<Assertion>`.
- MUST determine the original `<AddEntityRequest>` or `<AddKnownEntityRequest>` message to which the returned identifier corresponds.
- MUST use the appropriate federated name identifier for the user to obtain an identity token from the IDP for the invited user - this identity token targeted for itself.
- MUST, unless the SP did not include a `<Subscription>` in its `<AddEntityRequest>` message, obtain an identity token from the IDP for the invited user targeted at the SP.

  MUST forward on the identity token just received from the IDP in a `<Notify>` message, specifying the `SubscriptionID` of the previous `<Subscription>` element.
- MUST redirect the invited user's agent to the address previously specified by the `<PStoSPRedirectURL>` element in the original `<AddEntityRequest>` or `<AddKnownEntityRequest>` message.

Once the invited user has been redirected to the `<PStoSPRedirectURL>` address, the SP:

- MAY choose to send a `<samlp:AuthnRequest>` message to the IDP asking for a `<saml:AuthnStatement>` attesting to that user's authentication status there. In its `<Response>`, the IDP MUST use the same subject identifier for this `<saml:Assertion>` as previously delivered to the SP within the identity token through the 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:collect ion">
    <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>
```

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

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

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

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

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

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

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

11. PSa returns "true" to SPa.

```
<TestMembershipResponse>
  <Status code="OK"/>
  <TestResult>true</TestResult>
</TestMembershipResponse>
```

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

5.3. Group Operation

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

1. Alice signs on to SPa.

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

3. SPa/WSCa finds PSa, via DSa.

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

```

<ListMembersRequest>
  <TargetObjectID>https://ps.com/nmerflas</TargetObjectID>
</ListMembersRequest>

```

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

```

<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>
</ListMembersResponse>

```

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

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

```

<ResolveIdentifierRequest>
  <ResolveInput reqID="0">
    <TargetObjectID>https://psa.com/sdfhgusfsf</TargetObjectID> <!-- Bob -->
  </ResolveInput>
  <ResolveInput reqID="1">
    <TargetObjectID>https://psa.com/itndojd</TargetObjectID> <!-- Alice -->
  </ResolveInput>
</ResolveIdentifierRequest>

```

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

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

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

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

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

```
<ResolveIdentifierResponse>
  <Status code="OK"/>
  <ResolveOutput reqRef="0">
    <Token>
      <saml:Assertion>
       an identity token for Bob in SPa/WSCa’s namespace goes here
      </saml:Assertion>
    </Token>
  </ResolveOutput>
  <ResolveOutput reqRef="1">
    <Token>
      <saml:Assertion>
       an identity token for Alice in SPa/WSCa’s namespace goes here
      </saml:Assertion>
    </Token>
  </ResolveOutput>
</ResolveIdentifierResponse>
```

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

6. Security Considerations

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

- The header blocks specified in this document should be integrity-protected using the mechanisms detailed in [LibertySecMech].
- Header blocks should be signed in accordance with [LibertySecMech]. The receiver of a message containing a signature that covers specific header blocks should verify the signature as part of verifying the integrity of the header block.
- Metadata [LibertyMetadata] should be used to the greatest extent possible to verify message sender identity claims.
- Message senders and receivers should be authenticated to one another via the mechanisms discussed in [LibertySecMech].

7. XML Schema for ID-WSF People Service

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

```
<xs:schema
   targetNamespace="urn:liberty:ps:2006-08"
   xmlns="urn:liberty:ps:2006-08"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:ims="urn:liberty:ims:2006-08"
   xmlns:subs="urn:liberty:ssos:2006-08"
   xmlns:sec="urn:liberty:security:2006-08"
   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

   <xs:import namespace="urn:liberty:util:2006-08"
      schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
   <xs:import namespace="urn:liberty:ims:2006-08"
      schemaLocation="liberty-idwsf-idmapping-svc-v2.0.xsd"/>
   <xs:import namespace="urn:liberty:ssos:2006-08"
      schemaLocation="liberty-idwsf-subs-v1.0.xsd"/>
   <xs:import namespace="urn:liberty:security:2006-08"
      schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>
   <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
      schemaLocation="saml-schema-protocol-2.0.xsd"/>

   <!-- Definition of LocalizedDisplayNameType -->
   <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="xs:boolean" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <!-- Definition of TagType -->
   <xs:complexType name="TagType">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="Ref" type="xs:anyURI" use="required"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <!-- Declaration of ObjectID element -->
   <xs:element name="ObjectID" type="ObjectIDType"/>

   <!-- Declaration of TargetObjectID element -->
   <xs:element name="TargetObjectID" type="ObjectIDType"/>

   <!-- Definition of ObjectIDType -->
   <xs:complexType name="ObjectIDType">
      <xs:simpleContent>
        <xs:extension base="xs:anyURI"/>
      </xs:simpleContent>
   </xs:complexType>

   <!-- Declaration of Object element -->
   <xs:element name="Object" type="ObjectType"/>

   <!-- Definition of 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>

      <!-- Declaration of PStoSPRedirectURL-->

      <xs:element name="PStoSPRedirectURL" type="PStoSPRedirectURLType"/>

      <!-- Definition of PStoSPRedirectURLType-->

      <xs:complexType name="PStoSPRedirectURLType">
         <xs:annotation>
            <xs:documentation>
               When sending a AddEntityRequest to a PS provider,
               the SP may insert a PStoSPRedirectURL. It will be
               to this URL that the invited principals will be
               sent after federating their IDP account to the PS
               provider.
            </xs:documentation>
         </xs:annotation>
         <xs:simpleContent>
            <xs:extension base="xs:anyURI"/>
         </xs:simpleContent>
      </xs:complexType>

      <!-- Declaration of SPtoPSRedirectURL-->

      <xs:element name="SPtoPSRedirectURL" type="SPtoPSRedirectURLType"/>

      <!-- Definition of SPtoPSRedirectURLType-->

      <xs:complexType name="SPtoPSRedirectURLType">
         <xs:annotation>
            <xs:documentation>
               A PS provider may insert a SPtoPSRedirectURL in its
               AddEntityResponse. It will be to this URL that the
               invited principal will be sent after responding to the
               invitation.
            </xs:documentation>
         </xs:annotation>
         <xs:simpleContent>
            <xs:extension base="xs:anyURI"/>
         </xs:simpleContent>
      </xs:complexType>

      <!-- Declaration of QueryString -->

      <xs:element name="QueryString" type="QueryStringType"/>

      <!-- Definition of QueryStringType-->

      <xs:complexType name="QueryStringType">
         <xs:annotation>
            <xs:documentation>
               A PS provider may insert a QueryString in its
               AddEntityResponse or AddKnownEntityResponse. The
               invited Principal can present this artifact string
               to a certain provider.
            </xs:documentation>
         </xs:annotation>
         <xs:simpleContent>
            <xs:extension base="xs:string"/>
         </xs:simpleContent>
      </xs:complexType>

      <!-- Declaration of CreatePSObject element -->
      <xs:element name="CreatePSObject"/>

      <!-- Definition of RequestAbstractType -->
      <xs:complexType name="RequestAbstractType" abstract="true">
         <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:complexType>

      <!-- Definition of ResponseAbstractType -->
      <xs:complexType name="ResponseAbstractType" abstract="true">
       <xs:sequence>
            <xs:element ref="lu:Status"/>
       </xs:sequence>
         <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:complexType>

      <!-- Declaration of AddEntityRequest element -->
      <xs:element name="AddEntityRequest" type="AddEntityRequestType"/>
      <!-- Definition of AddEntityRequestType -->
      <xs:complexType name="AddEntityRequestType">
         <xs:complexContent>
            <xs:extension base="RequestAbstractType">
               <xs:sequence>
                  <xs:element ref="Object"/>
                  <xs:element ref="PStoSPRedirectURL" minOccurs="0"/>
                  <xs:element ref="CreatePSObject" minOccurs="0"/>
                  <xs:element ref="Subscription" minOccurs="0"/>
                  <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddEntityResponse element -->
      <xs:element name="AddEntityResponse" type="AddEntityResponseType"/>
      <!-- Definition of AddEntityResponseType -->
      <xs:complexType name="AddEntityResponseType">
         <xs:complexContent>
            <xs:extension base="ResponseAbstractType">
               <xs:sequence>
                  <xs:element ref="Object" minOccurs="0"/>
                  <xs:element ref="SPtoPSRedirectURL" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddKnownEntityRequest element -->
      <xs:element name="AddKnownEntityRequest" type="AddKnownEntityRequestType"/>
      <!-- Definition of AddKnownEntityRequestType -->
      <xs:complexType name="AddKnownEntityRequestType">
         <xs:complexContent>
            <xs:extension base="RequestAbstractType">
               <xs:sequence>
                  <xs:element ref="Object"/>
                  <xs:element ref="sec:Token"/>
                  <xs:element ref="CreatePSObject" minOccurs="0"/>
                  <xs:element ref="Subscription" minOccurs="0"/>
                  <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddKnownEntityResponse element -->
      <xs:element name="AddKnownEntityResponse" type="AddKnownEntityResponseType"/>
      <!-- Definition of AddKnownEntityResponseType -->
      <xs:complexType name="AddKnownEntityResponseType">
         <xs:complexContent>
            <xs:extension base="ResponseAbstractType">
               <xs:sequence>
                  <xs:element ref="Object" minOccurs="0"/>
                  <xs:element ref="SPtoPSRedirectURL" minOccurs="0" maxOccurs="1"/>
                  <xs:element ref="QueryString" minOccurs="0" maxOccurs="1"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddCollectionRequest element -->
      <xs:element name="AddCollectionRequest" type="AddCollectionRequestType"/>
      <!-- Definition of AddCollectionRequestType -->
      <xs:complexType name="AddCollectionRequestType" >
         <xs:complexContent>
            <xs:extension base="RequestAbstractType">
               <xs:sequence>
                  <xs:element ref="Object"/>
                  <xs:element ref="Subscription" minOccurs="0"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddCollectionResponse element -->
      <xs:element name="AddCollectionResponse" type="AddCollectionResponseType"/>
      <!-- Definition of AddCollectionResponseType -->
      <xs:complexType name="AddCollectionResponseType">
         <xs:complexContent>
            <xs:extension base="ResponseAbstractType">
               <xs:sequence>
                  <xs:element ref="Object" minOccurs="0"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddToCollectionRequest element -->
      <xs:element name="AddToCollectionRequest" type="AddToCollectionRequestType"/>
      <!-- Definition of AddToCollectionRequestType -->
      <xs:complexType name="AddToCollectionRequestType">
         <xs:complexContent>
            <xs:extension base="RequestAbstractType">
               <xs:sequence>
                  <xs:element ref="TargetObjectID"/>
                  <xs:element ref="ObjectID" minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element ref="Subscription" minOccurs="0"/>
               </xs:sequence>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>

      <!-- Declaration of AddToCollectionResponse element -->
      <xs:element name="AddToCollectionResponse" type="ResponseAbstractType"/>

      <!-- Declaration of RemoveEntityRequest element -->
      <xs:element name="RemoveEntityRequest" type="RemoveEntityRequestType"/>
      <!-- Definition of RemoveEntityRequestType -->
      <xs:complexType name="RemoveEntityRequestType">
```

```
      <xs:complexContent>
         <xs:extension base="RequestAbstractType">
            <xs:sequence>
               <xs:element ref="TargetObjectID" maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of RemoveEntityResponse element -->
   <xs:element name="RemoveEntityResponse" type="ResponseAbstractType"/>

   <!-- 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>

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

   <!-- Declaration of RemoveFromCollectionRequest element -->
   <xs:element name="RemoveFromCollectionRequest" type="RemoveFromCollectionRequestType"/>
   <!-- Definition of RemoveFromCollectionRequestType -->
   <xs:complexType name="RemoveFromCollectionRequestType">
      <xs:complexContent>
         <xs:extension base="RequestAbstractType">
            <xs:sequence>
               <xs:element ref="TargetObjectID"/>
               <xs:element ref="ObjectID" maxOccurs="unbounded"/>
               <xs:element ref="Subscription" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of RemoveFromCollectionResponse element -->
   <xs:element name="RemoveFromCollectionResponse" type="ResponseAbstractType"/>

   <!-- Declaration of ListMembersRequest element -->
   <xs:element name="ListMembersRequest" type="ListMembersRequestType"/>
   <!-- Definition of ListMembersRequestType -->
   <xs:complexType name="ListMembersRequestType">
      <xs:complexContent>
         <xs:extension base="RequestAbstractType">
            <xs:sequence>
               <xs:element ref="TargetObjectID" minOccurs="0"/>
               <xs:element ref="Subscription" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="Structured" type="xs:anyURI" use="optional"/>
            <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>

   <!-- Declaration of ListMembersResponse element -->
   <xs:element name="ListMembersResponse" type="ListMembersResponseType"/>
   <!-- Definition of ListMembersResponseType -->
   <xs:complexType name="ListMembersResponseType">
```

```
      <xs:complexContent>
         <xs:extension base="ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- 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>

   <!-- Declaration of QueryObjectsResponse element -->
   <xs:element name="QueryObjectsResponse" type="QueryObjectsResponseType"/>
   <!-- Definition of QueryObjectsResponseType -->
   <xs:complexType name="QueryObjectsResponseType">
      <xs:complexContent>
         <xs:extension base="ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of GetObjectInfoRequest element -->
   <xs:element name="GetObjectInfoRequest" type="GetObjectInfoRequestType"/>
   <!-- Definition of GetObjectInfoRequestType -->
   <xs:complexType name="GetObjectInfoRequestType">
      <xs:complexContent>
         <xs:extension base="RequestAbstractType">
            <xs:sequence>
               <xs:element ref="TargetObjectID" minOccurs="0"/>
               <xs:element ref="Subscription" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of GetObjectInfoResponse element -->
   <xs:element name="GetObjectInfoResponse" type="GetObjectInfoResponseType"/>
   <!-- Definition of GetObjectInfoResponseType -->
   <xs:complexType name="GetObjectInfoResponseType">
      <xs:complexContent>
         <xs:extension base="ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Object" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of SetObjectInfoRequest element -->
   <xs:element name="SetObjectInfoRequest" type="SetObjectInfoRequestType"/>
```

```
<!-- Definition of SetObjectInfoRequestType -->
<xs:complexType name="SetObjectInfoRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Object" maxOccurs="unbounded"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

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

<!-- Declaration of TestMembershipRequest element -->
<xs:element name="TestMembershipRequest" type="TestMembershipRequestType"/>
<!-- Definition of TestMembershipRequestType -->
<xs:complexType name="TestMembershipRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="TargetObjectID" minOccurs="0"/>
            <xs:element ref="sec:Token"/>
            <xs:element ref="Subscription" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<!-- Definition of ResultType -->
<xs:complexType name="ResultType">
   <xs:simpleContent>
      <xs:extension base="xs:boolean"/>
   </xs:simpleContent>
</xs:complexType>

<!-- Declaration of TestMembershipResponse element -->
<xs:element name="TestMembershipResponse" type="TestMembershipResponseType"/>
<!-- Definition of TestMembershipResponseType -->
<xs:complexType name="TestMembershipResponseType">
   <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
         <xs:sequence>
            <xs:element name="Result" type="ResultType" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<!-- Declaration of ResolveIdentifierRequest element -->
<xs:element name="ResolveIdentifierRequest" type="ResolveIdentifierRequestType"/>
<!-- Definition of ResolveIdentifierRequestType -->
<xs:complexType name="ResolveIdentifierRequestType">
   <xs:complexContent>
      <xs:extension base="RequestAbstractType">
         <xs:sequence>
            <xs:element ref="ResolveInput" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<!-- Declaration of ResolveInput element -->
<xs:element name="ResolveInput" type="ResolveInputType"/>
<!-- Definition of ResolveInputType -->
<xs:complexType name="ResolveInputType">
```

```
      <xs:complexContent>
         <xs:extension base="ims:MappingInputType">
            <xs:sequence>
               <xs:element ref="TargetObjectID" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of ResolveIdentifierResponse element -->
   <xs:element name="ResolveIdentifierResponse" type="ResolveIdentifierResponseType"/>
   <!-- Definition of ResolveIdentifierResponseType -->
   <xs:complexType name="ResolveIdentifierResponseType">
      <xs:complexContent>
         <xs:extension base="ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="ResolveOutput" maxOccurs="unbounded"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of ResolveOutput element -->
   <xs:element name="ResolveOutput" type="ims:MappingOutputType"/>

   <!-- Declaration of Subscription element -->
   <xs:element name="Subscription" type="subs:SubscriptionType"/>

   <!-- Declaration of Notification element -->
   <xs:element name="Notification" type="NotificationType"/>
   <!-- Definition of NotificationType -->
   <xs:complexType name="NotificationType">
     <xs:complexContent>
       <xs:extension base="subs:NotificationType">
         <xs:sequence>
           <xs:element ref="ItemData" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
       </xs:extension>
     </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of ItemData element -->
   <xs:element name="ItemData" type="ItemDataType"/>
   <!-- Definition of ItemDataType -->
   <xs:complexType name="ItemDataType">
     <xs:choice>
       <xs:element ref="Object" minOccurs="0" maxOccurs="unbounded"/>
       <xs:element ref="sec:Token" minOccurs="0"/>
     </xs:choice>
   </xs:complexType>

   <!-- Declaration of Notify element -->
   <xs:element name="Notify" type="NotifyType"/>
   <!-- Definition of NotifyType -->
   <xs:complexType name="NotifyType">
     <xs:complexContent>
       <xs:extension base="RequestAbstractType">
         <xs:sequence>
           <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
       </xs:extension>
     </xs:complexContent>
   </xs:complexType>

   <!-- Declaration of NotifyResponse element -->
   <xs:element name="NotifyResponse" type="subs:NotifyResponseType"/>
```

```
</xs:schema>
```

8. Abstract WSDL

```

<definitions
   name="id-wsf-ps_2006-08_wsdl_interface"
   targetNamespace="urn:liberty:ps:2006-08"
   xmlns:tns="urn:liberty:ps:2006-08"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
   xmlns:ps="urn:liberty:ps:2006-08"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
      http://schemas.xmlsoap.org/wsdl/
      http://www.w3.org/2006/02/addressing/wsdl
      http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

   <types>
      <xsd:schema>
         <xsd:import namespace="urn:liberty:ps:2006-08"
         schemaLocation="liberty-idwsf-people-service-v1.0.xsd"/>
      </xsd:schema>
   </types>

   <!-- Messages -->

   <!-- Adding a User -->

   <message name="AddEntityRequest">
      <part name="body" element="ps:AddEntityRequest"/>
   </message>

   <message name="AddEntityResponse">
      <part name="body" element="ps:AddEntityResponse"/>
   </message>

   <!-- Adding a Known User -->

   <message name="AddKnownEntityRequest">
      <part name="body" element="ps:AddKnownEntityRequest"/>
   </message>

   <message name="AddKnownEntityResponse">
      <part name="body" element="ps:AddKnownEntityResponse"/>
   </message>


   <!-- Removing a User -->

   <message name="RemoveEntityRequest">
      <part name="body" element="ps:RemoveEntityRequest"/>
   </message>

   <message name="RemoveEntityResponse">
      <part name="body" element="ps:RemoveEntityResponse"/>
   </message>

   <!-- Adding a Group -->

   <message name="AddCollectionRequest">
      <part name="body" element="ps:AddCollectionRequest"/>
   </message>

   <message name="AddCollectionResponse">
      <part name="body" element="ps:AddCollectionResponse"/>
   </message>
```

```

<!-- Removing a Group -->

<message name="RemoveCollectionRequest">
   <part name="body" element="ps:RemoveCollectionRequest"/>
</message>

<message name="RemoveCollectionResponse">
   <part name="body" element="ps:RemoveCollectionResponse"/>
</message>

<!-- Adding to a Group -->

<message name="AddToCollectionRequest">
   <part name="body" element="ps:AddToCollectionRequest"/>
</message>

<message name="AddToCollectionResponse">
   <part name="body" element="ps:AddToCollectionResponse" />
</message>

<!-- Removing From a Group -->

<message name="RemoveFromCollectionRequest">
   <part name="body" element="ps:RemoveFromCollectionRequest"/>
</message>

<message name="RemoveFromCollectionResponse">
   <part name="body" element="ps:RemoveFromCollectionResponse"/>
</message>

<!-- Listing Members -->

<message name="ListMembersRequest">
   <part name="body" element="ps:ListMembersRequest"/>
</message>

<message name="ListMembersResponse">
   <part name="body" element="ps:ListMembersResponse"/>
</message>

<!-- Retrieving Object Info -->

<message name="GetObjectInfoRequest">
   <part name="body" element="ps:GetObjectInfoRequest"/>
</message>

<message name="GetObjectInfoResponse">
   <part name="body" element="ps:GetObjectInfoResponse"/>
</message>

<!-- Updating Object Info -->

<message name="SetObjectInfoRequest">
   <part name="body" element="ps:SetObjectInfoRequest"/>
</message>

<message name="SetObjectInfoResponse">
   <part name="body" element="ps:SetObjectInfoResponse"/>
</message>

<!-- Querying Objects -->

<message name="QueryObjectsRequest">
   <part name="body" element="ps:QueryObjectsRequest"/>
</message>

```

```
<message name="QueryObjectsResponse">
   <part name="body" element="ps:QueryObjectsResponse"/>
</message>

<!-- Testing Membership -->

<message name="TestMembershipRequest">
   <part name="body" element="ps:TestMembershipRequest"/>
</message>

<message name="TestMembershipResponse">
   <part name="body" element="ps:TestMembershipResponse"/>
</message>

<!-- Resolving Identifiers -->

<message name="ResolveIdentifierRequest">
   <part name="body" element="ps:ResolveIdentifierRequest"/>
</message>

<message name="ResolveIdentifierResponse">
   <part name="body" element="ps:ResolveIdentifierResponse"/>
</message>

<!-- Port Type -->

<portType name="LibertyPS1">

   <operation name="AddEntity">
      <input message="tns:AddEntityRequest"
         wsaw:Action="urn:liberty:ps:2006-08:AddEntityRequest"/>
      <output message="tns:AddEntityResponse"
         wsaw:Action="urn:liberty:ps:2006-08:AddEntityResponse"/>
   </operation>

   <operation name="AddKnownEntity">
      <input message="tns:AddKnownEntityRequest"
         wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityRequest"/>
      <output message="tns:AddKnownEntityResponse"
         wsaw:Action="urn:liberty:ps:2006-08:AddKnownEntityResponse"/>
   </operation>

   <operation name="RemoveEntity">
      <input message="tns:RemoveEntityRequest"
         wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityRequest"/>
      <output message="tns:RemoveEntityResponse"
         wsaw:Action="urn:liberty:ps:2006-08:RemoveEntityResponse"/>
   </operation>

   <operation name="AddCollection">
      <input message="tns:AddCollectionRequest"
         wsaw:Action="urn:liberty:ps:2006-08:AddCollectionRequest"/>
      <output message="tns:AddCollectionResponse"
         wsaw:Action="urn:liberty:ps:2006-08:AddCollectionResponse"/>
   </operation>

   <operation name="RemoveCollection">
      <input message="tns:RemoveCollectionRequest"
         wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionRequest"/>
      <output message="tns:RemoveCollectionResponse"
         wsaw:Action="urn:liberty:ps:2006-08:RemoveCollectionResponse"/>
   </operation>

   <operation name="AddToCollection">
      <input message="tns:AddToCollectionRequest"
         wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionRequest"/>
      <output message="tns:AddToCollectionResponse"
```

```
           wsaw:Action="urn:liberty:ps:2006-08:AddToCollectionResponse"/>
     </operation>

     <operation name="RemoveFromCollection">
        <input message="tns:RemoveFromCollectionRequest"
           wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionRequest"/>
        <output message="tns:RemoveFromCollectionResponse"
           wsaw:Action="urn:liberty:ps:2006-08:RemoveFromCollectionResponse"/>
     </operation>

     <operation name="ListMembersOfCollection">
        <input message="tns:ListMembersRequest"
           wsaw:Action="urn:liberty:ps:2006-08:ListMembersRequest"/>
        <output message="tns:ListMembersResponse"
           wsaw:Action="urn:liberty:ps:2006-08:ListMembersResponse"/>
     </operation>

     <operation name="GetObjectInfo">
        <input message="tns:GetObjectInfoRequest"
           wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoRequest"/>
        <output message="tns:GetObjectInfoResponse"
           wsaw:Action="urn:liberty:ps:2006-08:GetObjectInfoResponse"/>
     </operation>

     <operation name="SetObjectInfo">
        <input message="tns:SetObjectInfoRequest"
           wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoRequest"/>
        <output message="tns:SetObjectInfoResponse"
           wsaw:Action="urn:liberty:ps:2006-08:SetObjectInfoResponse"/>
     </operation>

     <operation name="QueryObjects">
        <input message="tns:QueryObjectsRequest"
           wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsRequest"/>
        <output message="tns:QueryObjectsResponse"
           wsaw:Action="urn:liberty:ps:2006-08:QueryObjectsResponse"/>
     </operation>

     <operation name="TestMembership">
        <input message="tns:TestMembershipRequest"
           wsaw:Action="urn:liberty:ps:2006-08:TestMembershipRequest"/>
        <output message="tns:TestMembershipResponse"
           wsaw:Action="urn:liberty:ps:2006-08:TestMembershipResponse"/>
     </operation>

     <operation name="ResolveIdentifier">
        <input message="tns:ResolveIdentifierRequest"
           wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierRequest"/>
        <output message="tns:ResolveIdentifierResponse"
           wsaw:Action="urn:liberty:ps:2006-08:ResolveIdentifierResponse"/>
     </operation>

  </portType>

<!--
An example of a binding and service that can be used with this
abstract service description is provided below.
-->

  <binding name="PeopleServiceSoapBinding" type="tns:LibertyPS1">

     <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

     <operation name="AddEntity">
        <soap:operation/>
        <input>  <soap:body use="literal"/> </input>
```

```
            <output> <soap:body use="literal"/> </output>
        </operation>

        <operation name="AddKnownEntity">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="RemoveEntity">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="AddCollection">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="RemoveCollection">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="AddToCollection">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="RemoveFromCollection">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="ListMembersOfCollection">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="GetObjectInfo">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="SetObjectInfo">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="QueryObjects">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="TestMembership">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>
        <operation name="ResolveIdentifier">
            <input>  <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
        </operation>

    </binding>

    <service name="PeopleService">
        <port name="PeoplePort" binding="ps:PeopleServiceSoapBinding">

            <!-- Modify with the REAL SOAP endpoint -->

            <soap:address location="http://example.com/peopleservice"/>
        </port>
    </service>

</definitions>
```

References

Normative

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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