Liberty ID-WSF Subscriptions and Notifications

Version: 1.0

Editors:
Sampo Kellomäki, Symlabs, Inc.

Contributors:
Robert Aarts, Hewlett-Packard
Rajeev Angal, Sun Microsystems, Inc.
Conor Cahill, America Online, Inc.
Carolina Canales-Valenzuela, Ericsson
Darryl Champagne, IEEE-ISTO
Andy Feng, America Online, Inc.
Gael Gourmelen, France Télécom
Jeff Hodges, NeuStar, Inc.
Jukka Kainulainen, Nokia Corp.
Lena Kannappan, France Télécom
John Kemp, Nokia Corporation
Rob Lockhart, IEEE-ISTO
Paul Madsen, NTT
Aravindan Ranganathan, Sun Microsystems, Inc.
Matti Saarenpää, Nokia Corporation
Jonathan Sergent, Sun Microsystems, Inc.
Lakshmanan Suryanarayanan, America Online, Inc
Greg Whitehead, Hewlett-Packard

Abstract:

This specification provides protocols for subscription and notification. A *subscription* is a mechanism by which a WSC can register to receive *notifications* from a WSP when some data changes or some event happens. The subscriptions and notifications are applicable to any ID-WSF-based service, but specific guidance is provided on how to apply them on a DST-based service.

Filename: liberty-idwsf-subs-v1.0.pdf

Notice

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

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

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

Contents

1. Introduction

This specification provides protocols for subscription and notification. A *subscription* is a mechanism by which a WSC can register to receive *notifications* from a WSP when some data changes or some event happens.

Since there is usually data involved, it is common that data services, based on the Liberty ID-WSF Data Services Template [LibertyDST], will incorporate subscription features. A fair amount of this specification is dedicated to these situations, including subscription as a side effect of query or create, and subscription by explicit manipulation of subscription objects, using a DST-derived interface.

However, subscriptions can profitably be employed even outside data services and there is no need to base a service on DST for it to use subscriptions. The Liberty ID-WSF People Service Specification [LibertyPeopleService] illustrates this approach. In such case, the service in question is responsible for providing the methods for subscription and subscription management, while using **<Subscription>** element as defined in this document.

1.1. Notation

When capitalized, the key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this specification are to be interpreted as described in [RFC2119]. When these words are not capitalized, they are meant in their natural-language sense.

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

1.2. Liberty Considerations

This specification contains enumerations of values that are centrally administered by the Liberty Alliance Project. Although this document may contain an initial enumeration of approved values, implementers of the specification MUST implement the list of values whose location is currently specified in [LibertyReg] according to any relevant processing rules in both this specification and [LibertyReg].

1.3. Namespaces

The namespaces described in Table 1 are used.

Table 1. Normatively referenced XML namespaces

Prefix	URI	Description
subs:	urn:liberty:ssos:2006-08	Target namespace of Subscriptions and Notifications schema.
subsref:	urn:liberty:ssos:2006-08:ref	Target namespace of Subscriptions and Notifications reference model.
dst:	urn:liberty:dst:2006-08	Target namespace of DST utility schema.
dstref:	urn:liberty:dst:2006-08:ref	Target namespace of DST reference model.
xml:	http://www.w3.org/XML/1998/namespace	W3C XML [XML].
xs:	http://www.w3.org/2001/XMLSchema	W3C XML Schema Definition Language [Schema1-2].
ds:	urn:liberty:disco:2006-08	Liberty ID-WSF Discovery Service [LibertyDisco].
sb:	urn:liberty:sb:2006-08	Liberty ID-WSF SOAP Binding Extension [LibertySOAPBinding].

1.4. Applying Subscriptions to DST-Based Services

While Subscriptions and Notifications can be adopted by any ID-WSF-based service, if these features are to be adopted by a DST-based service, it SHOULD be done on the basis of the extended reference model described in Section 8.1, i.e., this model should replace the model specified in [LibertyDST] Section 11.1 "DST Reference Model Schema."

Specifically, the service is expected to provide definitions for

1. **<Notify>** and **<NotifyResponse>**. In particular, the definition of **<Notify>** will depend on the data model and schema adopted by the service.

2. **<Subscription>**, to incorporate the aspects of service-dependent query language.

3. An interface for manipulation of subscription objects. In the DST model, the regular "CRUD" interface is used with the special object type "_Subscription." A service that is not otherwise DST-based may wish to support the DST interface just for subscription object manipulation. Providing this facility is optional for a service specification.

4. A means for establishment of subscriptions as a side effect ("piggy-backed") of other operations. Providing this facility is optional for a service specification. An example of how this could be accomplished as a side effect of **<Create>** is provided in Section 3.2.

2. General Rules Regarding Subscriptions and Notifications

2.1. Second Level <Status> Codes for Subscriptions

The following second level status codes are defined for subscriptions:

```
EmbeddedSubscriptionsNotSupported
InvalidSubscriptionID
MissingSubscriptionID
```

If a request or notification fails for some reason, the `ref` XML attribute of the <**Status**> element SHOULD contain the value of the `itemID` XML attribute of the offending element in the request message. Subscription and notifications messages use `subscriptionID` XML attributes instead of `itemID` XML attributes and those should be used when reporting failure statuses related to the sub-elements of subscription and notification messages. When the offending element does not have the `itemID` or `subscriptionID` XML attribute, the reference SHOULD be made using the value of the `id` XML attribute, if that is present.

If it is not possible to refer to the offending element (as it has no `id`, `itemID`, or `subscriptionID` XML attribute), the reference SHOULD be made to the ancestor element having a proper identifier XML attribute closest to the offending element.

Since both `itemID` and `subscriptionID` can be used to refer to a failed element, the two IDs form one namespace. Care should be taken to avoid `id` values that would create ambiguity.

2.2. Discovery Option Keywords for Indicating Lack of Subscription Support

A WSP MAY register the following discovery option keywords to indicate that it does not support certain types of subscription manipulations:

```
urn:liberty:subs:noSubscribe
urn:liberty:subs:noQuerySubscriptions
```

2.3. CRUD Manipulation Using Object Type "_Subscription"

Service specifications that support subscriptions must use *object type* "_Subscription" to designate them.

As a service may support different types of objects, the `SelectType` MUST be defined so that it supports all different types of objects supported by the service, including "_Subscription."

If a service supports subscriptions, the `SelectType` MUST be specified so that it can carry strings containing XPath expressions. If the same service type supports objects which do not use XPath but e.g., own special element structure, the `SelectType` MUST still make it possible to carry just strings, this might require specifying `mixed="true"`, but a service type MUST NOT use real mixed type and have strings and elements at the same time, so either strings or sub-elements are allowed, but not both at the same time.

2.4. No itemIDRef for Subscription-Related <ItemData>

The <**ItemData**> elements returning changed expiration times for subscriptions created based on the request message MUST NOT contain any `itemIDRef` XML attribute. They contain <**Subscription**> elements, which carry `subscriptionID` XML attributes (see Section 4).

3. Piggy-Backing Subscriptions to DST Operations

N.B. Subscription to <**Delete**> is generally not meaningful and is not discussed here.

3.1. <Query> with <Subscription>

While querying data, it is possible to simultaneously subscribe to future changes of that data by including <**Subscription**> elements inside the <**Query**> (see Section 4). These <**Subscription**> elements MUST refer to the <**QueryItem**> elements using <**RefItem**> elements to indicate that a WSC wants to subscribe to the same data it is querying. The <**Subscription**> elements MAY also have their own <**ResultQuery**> elements to define additional data to which a WSC wants to subscribe. A service specification and a WSP MAY specify additional restrictions on how subscriptions are supported inside queries, or that they are not supported at all.

```
<xs:complexType name="QueryType">
  <xs:complexContent>
    <xs:extension base="dst:RequestType">
      <xs:sequence>
        <xs:element ref="subsref:TestItem" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="subsref:QueryItem" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 1. Definition of Query that Supports Piggy-Backed Subscription

3.2. <Create> with <Subscription>

A <**Create**> element may also contain one or more <**Subscription**> elements to subscribe, for example, to future changes of the data just created (see Section 4).

```
<xs:complexType name="CreateType">
  <xs:complexContent>
    <xs:extension base="dst:RequestType">
      <xs:sequence>
        <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="subsref:CreateItem" minOccurs="1" maxOccurs="unbounded"/>
        <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 2. Definition of Create that Supports Piggy-Backed Subscription

The <**CreateResponse**> element contains, in addition to a <**Status**> element, possible <**ItemData**> elements, which carry requested data related to the data just created. <**ItemData**> elements may also carry information about subscriptions, when a WSP changed or added the expiration time. For example, returned data could include a unique ID assigned to the data object just created.

3.3. <Modify> with <Subscription>

A <**Modify**> may contain <**Subscription**> element(s) when a WSC wants to subscribe to the data it is modifying. These <**Subscription**> elements MUST refer to the <**ModifyItem**> elements using <**RefItem**> element(s). The <**Subscription**> elements MAY also have their own <**ResultQuery**> element(s) to define additional data to which a

WSC wants to subscribe. See Section 4 for more information. A service specification and a WSP MAY set additional restrictions, i.e., how subscriptions are supported inside modification requests, if the support is allowed at all.

A **<ModifyResponse>** may contain **<ItemData>** element(s). The elements can contain either data requested with **<ResultQuery>** elements or **<Subscription>** elements when a WSP has modified the expiration time.

```
<xs:complexType name="ModifyType">
  <xs:complexContent>
    <xs:extension base="dst:RequestType">
      <xs:sequence>
        <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="subsref:ModifyItem" minOccurs="1" maxOccurs="unbounded"/>
        <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 3. Definition of Modify that Supports Piggy-Backed Subscription

4. Subscriptions

The subscriptions are a mechanism through which WSCs can request notifications when a specified event happens. The basic case is subscribing to change notifications to get updates when the data hosted by a data service related to a Principal changes. A WSC may subscribe to change notifications even before the data exists. For example, a WSC may want to know when a Principal adds an email address to her profile. The change of data is not the only possible reason for a notification, there can be service-specific triggers for notifications, e.g., periodic notifications containing current values and notifications after a Principal switches on her terminal.

As the notifications reveal not only the data they are carrying, but also that a certain thing has just happened, WSPs must be very careful to make sure they honor the privacy of the Principals.

This document specifies one `objectType`, the "_Subscription." These can be accessed and manipulated like any other objects; they can be created, deleted, modified and queried. The difference from other object types is that "_Subscription" objects can be created by means other than with the normal **<Create>**. A **<Subscription>** element can be embedded within other request types to make it easier to subscribe to the data accessed with those requests. For example, a WSC may subscribe to the data it just modified with a **<Modify>**. This can be done by adding a **<Subscription>** element into the **<Modify>** request without a need to make a separate **<Create>** request to create a "_Subscription" object.

When subscriptions are supported in addition to creating them, deleting subscriptions with **<Delete>** MUST be supported. Renewing subscriptions by modifying the expiration time (`expires` XML attribute) using **<Modify>** SHOULD also be supported and modifying other parameters of subscriptions MAY be supported.

Notifications are carried inside **<Notify>** elements. The notifications are specified in Section 5.

4.1. <Subscription> element

The **<Subscription>** element contains all the parameters for a subscription. It defines what data a WSC wants to have, where it should be sent, when a subscription expires, which events should trigger notifications, etc.

```
<xs:complexType name="SubscriptionType">
  <xs:sequence>
    <xs:element ref="subs:RefItem" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="lu:Extension" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="subscriptionID" use="required" type="lu:IDType"/>
  <xs:attribute name="notifyToRef" use="required" type="xs:anyURI"/>
  <xs:attribute name="adminNotifyToRef" use="optional" type="xs:anyURI"/>
  <xs:attribute name="starts" use="optional" type="xs:dateTime"/>
  <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
  <xs:attribute name="id" use="optional" type="xs:ID"/>
  <xs:attribute name="includeData" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Yes"/>
        <xs:enumeration value="No"/>
        <xs:enumeration value="YesWithCommonAttributes"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:element name="RefItem" type="subs:RefItemType"/>
<xs:complexType name="RefItemType">
  <xs:attribute name="subscriptionID" use="optional" type="lu:IDType"/>
  <xs:attribute ref="lu:itemIDRef" use="required"/>
</xs:complexType>
```

Figure 4. Utility Schema for Subscription

```
<xs:element name="Subscription" type="subsref:SubscriptionType"/>
<xs:complexType name="SubscriptionType">
  <xs:complexContent>
    <xs:extension base="subs:SubscriptionType">
      <xs:sequence>
        <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Aggregation" minOccurs="0" maxOccurs="1" type="subsref:AggregationType"/>
        <xs:element name="Trigger" minOccurs="0" maxOccurs="1" type="subsref:TriggerType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 5. Reference Model Definition of Subscription

The different subscriptions related to a same resource are distinguished from each other by Ids (`subscriptionID` XML attribute). The `subscriptionID` XML attribute MUST be unique within all subscriptions a WSC has at a WSP. A WSC specifies the value of the `subscriptionID` XML attribute of a subscription when creating a new subscription. After a subscription is accepted, it is referenced using this value and all notifications also carry the `subscriptionID` XML attribute.

4.2. Selecting Data to which a Subscription Applies

The first parameter inside the <**Subscription**> element is the <**ResultQuery**> element. This is the basic data selection element used in multiple places. It defines what data a notification should return. The use of the <**ResultQuery**> element inside the <**Subscription**> element might be a bit different than its use when querying and modifying. The specifications for services MUST specify possible differences. Different parameters of the <**ResultQuery**> element are described together with processing rules in [LibertyDST] Section 4 "Querying Data." There can be more than one <**ResultQuery**> element inside a <**Subscription**> element.

The <**RefItem**> element is used when a WSC wants to subscribe to the data it is accessing with the *RequestElement*. Finally, a <**Trigger**> element can be used to specify arbitrary conditions for triggering notifications.

Normally, a notification is triggered when the data addressed by the <**ResultQuery**> or <**RefItem**> element has changed. There can also be other reasons that trigger notifications. The <**Trigger**> element contains those triggers. The <**Trigger**> element is of type `TriggerType`, which MUST be defined by the service's schema. The service specification MUST define semantics and values for this parameter. When the <**Trigger**> element is not used, a WSC requests normal change notifications unless otherwise specified by a service specification.

4.3. Providing Information for Sending Notifications

The XML attribute `notifyToRef` contains a reference to an endpoint object, defined in the SOAP headers of the message, which indicates where and how (e.g., using which security mechanism and credentials or tokens) the notification must be sent. The `notifyToRef` and `adminNotifyToRef` design patterns and the associated end point objects are further described in [LibertySOAPBinding].

If the `adminNotifyToRef` XML attribute is not specified, the subscription end notifications are sent to the end point indicated by the `notifyToRef` XML attribute. The purpose of the `adminNotifyToRef` XML attribute is to make it possible to receive notifications in one point and manage changes to subscriptions in another point.

There can be different types of notifications. For example, a notification can be sent immediately or multiple notifications can be sent in a bigger batch. The element <**Aggregation**> defines what type of notifications a WSC is requesting. The element <**Aggregation**> further describes, in a service-specification-dependent way, how the notifications are to be batched. It is of type `AggregationType`, which MUST be specified, including the detailed semantics and allowed values, by the service specification.

Usually, a notification contains data related to a resource. Sometimes, a notification could be used to indicate that an event related to a resource has happened, e.g., the data addressed by the <**ResultQuery**> element has changed without

reporting the changed data. The XML attribute `includeData` defines whether or not the data of the changed object should be included in the notification messages. Possible values are `Yes` (data is returned), `No` (no data is returned), and `YesWithCommonAttributes` (the data is returned with the common XML attributes). A service specification SHOULD specify a default value. It should be noted that sending just a change notification without any actual data usually has less security and privacy issues compared to cases when the data is also included in a notification message.

4.4. Expiration of Subscription

A subscription is not valid forever. The `starts` XML attribute defines the time after which a subscription is valid and notifications can be sent if the triggering event occurs. The `starts` XML attribute MUST be used only when a subscription is not valid immediately after processing the request. The `expires` XML attribute defines the time when a subscription expires, if not renewed before that time.

If credentials needed for subscription expire earlier than a subscription, and a WSC does not provide new credentials before they expire, the subscription MUST expire.

4.5. Common Processing Rules for Subscriptions

When subscriptions are requested by a WSC, the following processing rules MUST be obeyed. (Note: these rules are valid regardless of the way a subscription is requested.)

A subscription is one entity which either succeeds or fails. A subscription is identified with a `subscriptionID`.

4.5.1. General Processing Rules for Subscriptions

1. If a WSP fails to process the parameters of a subscription properly according to the specified rules, it MUST NOT accept that subscription and SHOULD use the appropriate second level status code to indicate the reason. One **<Subscription>** element in a request message may specify more than one subscription since **<RefItem>** elements may have their own `subscriptionID` XML attributes. The implication of this is that one **<Subscription>** element may contain subscriptions which succeed and subscriptions which fail. Failure of even one subscription SHOULD cause an error response unless the service specification specifies rules for partial success.

2. When subscriptions are created within **<Query>** or **<Modify>** or within **<Create>** such that they are direct child elements of the **<Create>** (referring to **<CreateItem>** elements), the failure to process those subscription or rejecting those subscriptions for other reasons (e.g., policies) is not considered as a failure of a **<Query>**, **<Modify>** or **<Create>** request. The normal **<Query>**, **<Modify>**, or **<Create>** parameters inside **<QueryItem>**, **<ModifyItem>**, or **<CreateItem>** elements, respectively, MUST be processed normally, even if a subscription referring to those fails, unless otherwise stated by a service specification. If a subscription is not accepted, a WSP MUST indicate this back to a WSC. For example, if a WSP does not support **<Subscription>** elements embedded as a direct child of a **<Query>**, a **<Modify>**, or a **<Create>** element and it receives such, it MUST use the second level status code `EmbeddedSubscriptionsNotSupported` to indicate this. If processing of an embedded **<Subscription>** element fails, the proper second level status code MUST be returned and the failed **<Subscription>** element MUST be referenced using the `subscriptionID` as the value of the `ref` XML attribute of the **<Status>** element. As failing embedded subscription does not cause failure of a request message, a WSC MUST check the returned second level status elements to find out whether those subscriptions were accepted by a WSP or not.

3. When a new subscription is created the way data objects are normally created (i.e., within **<NewData>** of a **<CreateItem>**), the normal processing rules MUST be applied with the exception that this specification gives some object-type-specific processing rules and more detailed status codes to be used, when applicable, instead of the generic `InvalidData`. When a WSP does not support subscriptions and a WSC tries to create one in the way data objects are created, it should return the second level status code `UnsupportedObjectType` when subscriptions are allowed for the service type but not supported by a WSP and `InvalidObjectType` when they are not allowed for the service type.

4. The values of the `subscriptionID` XML attributes are WSC-specific. When a new subscription is created, it MUST use a `subscriptionID` different from any other subscription the same WSC has at the same WSP. If a WSC tries to create a new subscription which has a conflicting `subscriptionID` value, a WSP MUST reject that and it SHOULD use the second level status code `InvalidSubscriptionID`.

5. An implementation MAY decompose a composite subscription object into unit subscriptions. For example, if a subscription object has multiple **<RefItem>** elements with different `subscriptionID` XML attributes, this is interpreted to create multiple logical subscriptions. An implementation may, indeed, choose to handle them as separate subscriptions. While an implementation MUST support the creation of composite subscriptions, it NEED NOT support composite subscriptions on **<Query>**, **<Modify>**, and **<Delete>** interfaces involving objects of type "_Subscription."

4.5.2. Processing Rules for Data to which the Subscription Applies

A WSC must specify in a subscription the data to which the subscription applies.

1. When **<Subscription>** elements contain a **<ResultQuery>** element, a WSP MUST process its content in a similar fashion as it processes the same parameters in the case of a normal query, taking into account that no data is returned immediately. A WSP MUST support the requested `objectType` and **<Select>**. If a WSP does not support sorting and it is requested by a WSC, a WSP SHOULD still accept the subscriptions and return data unsorted in notifications. The `changedSince` XML attribute MUST be ignored, if present. When notifications are expected to contain only the changed data, a WSC MAY use **<ChangeFormat>** to indicate formats it supports. Note that with subscriptions, the **<ChangeFormat>** is used without having the `changedSince` XML attribute (required in regular queries). The `predefined` XML attribute can be used instead of other parameters. See [LibertyDST], Section 3.7 "Selection" and Section 4.4 "Processing Rules for Queries" for more details and proper status codes.

2. When a **<RefItem>** element is included in a subscription, it MUST contain an `itemIDRef` XML attribute. The value of this XML attribute MUST be the same as the value of an `itemID` XML attribute of a **<QueryItem>**, a **<CreateItem>**, or a **<ModifyItem>**, depending on the message. This creates a subscription to all of the data manipulated in the referenced element.

3. If the value of the `itemIDRef` XML attribute does not match to any relevant `itemID`, the subscription MUST NOT be accepted and the second level status code `InvalidItemIDRef` SHOULD be used to indicate the reason.

4. If a **<RefItem>** element contains a `subscriptionID` XML attribute and it has a different value than the `subscriptionID` XML attribute of the **<Subscription>** element, the **<RefItem>** element defines a new subscription which inherits other parameters, except **<ResultQuery>** elements, `subscriptionID` XML attribute, and possible other **<RefItem>** elements from the **<Subscription>** element in which the **<RefItem>** element is contained. Each **<RefItem>** that has a `subscriptionID` XML attribute creates a new independent subscription. If multiple **<RefItem>** elements have the same value of the `subscriptionID`, they all form one subscription together and that subscription has multiple sets of selection parameters. If data selected by any of the sets is changed, a notification is sent.

5. A **<Subscription>** element may contain any number of **<ResultQuery>**, **<RefItem>**, and **<Trigger>** elements. If none of the elements **<ResultQuery>**, **<RefItem>** or **<Trigger>** are present, the processing of the **<Subscription>** element MUST fail unless the service specification has defined, what this kind of a case means, e.g., some default values are defined for parameters and those are used or a WSC subscribes to the whole resource. When the processing of a **<Subscription>** element fails due to not having **<ResultQuery>**, **<RefItem>** or **<Triggers>** present, the second level status code `MissingSelect` SHOULD be used to indicate this.

4.5.3. Processing Rules for <Aggregation> and <Trigger>

1. A WSP MUST follow the processing rules defined in the service specification for the elements <**Aggregation**> and <**Trigger**>. If the use of these elements is not specified for the service or specified, but not supported by a WSP, and either of both of them are included in a <**Subscription**> element in a <**Subscribe**> request, the processing of the <**Subscription**> MUST fail and a second level status code SHOULD be used, either `AggregationNotSupported` or `TriggerNotSupported`, to indicate this.

2. If a WSP does support aggregation, but not the type of <**Aggregation**> a WSC requests, the processing of the <**Subscription**> MUST fail and the second level status code `RequestedAggregationNotSupported` SHOULD be used, in addition to the top level status code, to indicate this. Similarly, if a WSP does support triggers, but not the type of a <**Trigger**> a WSC requests, the processing of the <**Subscription**> MUST fail and the second level status code `RequestedTriggerNotSupported` SHOULD be used, in addition to the top level status code, to indicate this.

4.5.4. Processing Rules for First Notification and Expiry of Subscription

A WSC may request when the first notification may be sent and when a subscription should expire.

1. If a <**Subscription**> element contains a `starts` XML attribute, subscription MUST be valid *only* after the time defined. If the `starts` XML attribute is omitted, the subscription MUST be valid immediately after processing the request. Also, if the time specified by the `starts` XML attribute is in the past, then that subscription, if accepted by a WSP, MUST be valid immediately after processing the request.

2. The time specified by the `expires` XML attribute MUST be the same time or a later time than the time specified by the `starts` XML attribute in the same <**Subscription**> element. It also MUST be later than the current time. If either of the checks is not passed, then the processing of the <**Subscription**> MUST fail and the second level status code `InvalidExpires` SHOULD be used, in addition to the top level status code, to indicate this.

3. A WSP MAY change the time when a subscription expires from the expiration time requested by a WSC with the `expires` XML attribute. A WSP MAY shorten the expiration time, but it MUST NOT make the expiration time longer. If no `expires` XML attribute is included in a <**Subscription**> element in a request from a WSC, a WSP MUST decide the expiration time for the subscription, if expiration times are required either by the service specification or the WSP. A WSP MUST return the expiration time in the response message if it is changed compared to what a WSC requested. This information is returned by returning a <**Subscription**> element with XML attributes `subscriptionID` and `expires` inside a <**Data**> element in the case of a <**QueryResponse**> and inside a <**ItemData**> in the case of a <**CreateResponse**> and <**ModifyResponse**>. That <**Data**> or <**ItemData**> element MUST NOT contain any other data than <**Subscription**> elements created based on one <**Subscription**> element or, when a normal data object creation method has been used, <**Subscription**> elements created with one <**CreateItem**> element. The <**Data**> or <**ItemData**> element SHOULD NOT contain any `itemIDRef` XML attributes. The matching is done based on the `subscriptionID` XML attributes carried inside <**Subscription**> elements.

4. If a WSC wants to renew an existing subscription before it has ended, it MUST modify that subscription and give a new value for the `expires` XML attribute of that subscription. A WSP MAY modify the new value in the same way as it MAY modify the proposed value for a new subscription.

5. There is one special case when using subscriptions expirations. When the `starts` and `expires` XML attributes have exactly the same value, the meaning is that a notification MUST be sent exactly at that time whether some event (e.g., data change) has happened or not. A WSC wants to get current values of the data (e.g., location) exactly at that time, even if the values have stayed the same for a long time (e.g., a Principal has not moved).

4.5.5. Processing Rules When the Access and Privacy Policies Forbid Subscription

The access and privacy policies specified by the resource owner may not allow a WSC to subscribe to the data of a resource or to some events related to a resource.

1. When a WSP processes a <**Subscription**> element, it MUST check whether the resource owner (the Principal, for example) has given consent to return the requested data and the fact that an event or data change has happened in notification messages. To be able to check WSC-specific access rights, the WSP MUST authenticate the WSC (see [LibertySecMech]). The WSP MUST also check that any usage directive given in the request is acceptable based on the usage directives defined by the resource owner (see [LibertySOAPBinding]). If either check fails, the WSP MUST NOT accept the subscription and the processing of that <**Subscription**> MUST fail. The WSP MAY try to get consent from the Principal while processing the request, perhaps by using an interaction service (see [LibertyInteract]). A WSP might check the access rights and policies in usage directives at a higher level, before getting to DST processing and MAY, in this case, just return an ID-* Fault Message (see [LibertySOAPBinding]) without processing the *RequestElement* element at all if the requesting WSC is not allowed to access the data in question.

2. Note that there can be consent for subscribing to some data element but not its XML attributes. A Principal might not want to release the `modifier` XML attribute if she does not want to reveal information about which services she uses. If a WSC is not allowed to get all the data, but some of the data it wants, a WSP SHOULD accept the subscription, but it MAY also reject it. If a subscription is accepted, the data for which there is no consent from the Principal MUST be handled as if there were no data. Also that data, or the fact that the data has changed, MUST NOT be included in the notification messages sent later on.

3. If a WSC has made a subscription and included the usage directive it has promised to obey, then later wants to change the usage directive, it MUST cancel the subscription and make a new subscription with the new value for the usage directive.

4.6. `SelectType` for Subscription Objects

> N.B. This subsection is about selecting the wanted subscription objects when deleting and modifying them, not about a subscription selecting the right data for notifications. When a WSC wants to access existing subscription objects after they have been created, it must be able to select the right ones. XPath is used to select the subscription objects.

The minimum a WSP MUST support is `/ns:Subscription[@ns:subscriptionID="xx"]` so that a WSC can delete an existing subscription using <**Delete**>. Of course, the `objectType` XML attribute must have the value "_Subscription." Just by setting the `objectType` XML attribute to "_Subscription," a WSC can delete all subscriptions it has related to a resource.

A WSP SHOULD also support `/ns:Subscription[@ns:subscriptionID="xx"]/@ns:expires` to make it possible to renew a subscription before it expires by using <**Modify**>.

A WSP MAY also support:

```
/ns:Subscription[@ns:subscriptionID="xx"]/ns:notifyToRef
/ns:Subscription[@ns:subscriptionID="xx"]/ns:adminNotifyToRef
```

to make it possible to change endpoints and related information.

A WSP MUST support abbreviated XPath, as described in [XPATH] Section 2.5.

A WSP MAY also support full XPath to make it possible to modify all the parameters of a subscription without the need to rewrite those parameters which do not change, but a subscription can be updated by selecting it using `/ns:Subscription[@ns:subscriptionID]` and rewriting the whole subscription.

4.7. Support for <Subscription> Conditioned by <TestItem>

A WSC can subscribe to be notified if the results of a test change. For example, if the original result of a test was true, the WSC can ask to be notified when the result becomes false and vice versa.

The WSC indicates that it is subscribing to the test results by specifying `itemIDRef` XML attribute that references the appropriate **<TestItem>** element. The result is reported via **<TestResult>** in the **<Notification>**.

1. A service specification MAY restrict, or forbid, use of **<TestItem>** in conjunction with **<Subscription>**. If use of **<TestItem>** is fully supported, the WSP MAY register the discovery option keyword

   ```
   urn:liberty:subs:contingentSubscription
   ```

2. A **<Subscription>** that references **<TestItem>** MUST NOT have **<Trigger>**. The only valid triggering condition is "on change," which is implied, thus no **<Trigger>** element is necessary.

3. If the `itemIDRef` attribute does not match a **<TestItem>**, then the WSP MUST stop processing the **<Subscription>** and return a second level status code `NoSuchTest`.

4. If **<Subscription>** has an `itemIDRef` XML attribute, the WSP MUST detect changes to the result of evaluation of the **<TestItem>** referenced by the `itemIDRef` and send notifications when such changes occur.

5. The scope of the `itemIDRef` is one **<Query>**, **<Create>**, or **<Modify>**. `itemIDRef` MUST NOT refer to an `itemID` in another top level element. The `itemID` XML attributes of **<TestItem>** elements MUST be unique within one **<Query>**, **<Create>**, or **<Modify>** element in the request. The **<TestItem>**, **<ResultQuery>**, and **<QueryItem>** share same `itemID` space.

5. Notifications

When a WSC has subscribed to some data or event, a WSP will send notifications when the subscribed data changes or the event happens. A notification can also be sent when a subscription expires or is changed by a WSP (e.g., it shortens the expiration time).

5.1. <Notify> Element

Notifications are carried by <**Notify**> elements. One <**Notify**> element may carry one or more <**Notification**> elements. Otherwise, the <**Notify**> element just has the normal `id` and `timestamp` XML attributes.

```
<xs:attributeGroup name="NotifyAttributeGroup">
  <xs:attribute name="timeStamp" use="optional" type="xs:dateTime"/>
</xs:attributeGroup>
<xs:complexType name="NotificationType">
  <xs:sequence>
    <xs:element ref="lu:TestResult" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" use="optional" type="xs:ID"/>
  <xs:attribute name="subscriptionID" use="required" type="lu:IDType"/>
  <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
  <xs:attribute name="endReason" use="optional" type="xs:anyURI"/>
</xs:complexType>
<xs:complexType name="NotifyResponseType">
  <xs:complexContent>
    <xs:extension base="lu:ResponseType"/>
  </xs:complexContent>
</xs:complexType>
```

Figure 6. Utility Schema for Notify

```
<xs:complexType name="NotifyType">
  <xs:complexContent>
    <xs:extension base="dst:RequestType">
      <xs:sequence>
        <xs:element ref="subsref:Notification" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Notification" type="subsref:NotificationType"/>
<xs:complexType name="NotificationType">
  <xs:complexContent>
    <xs:extension base="subs:NotificationType">
      <xs:sequence>
        <xs:element ref="subsref:ItemData" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="NotifyResponseType">
  <xs:complexContent>
    <xs:extension base="subs:NotifyResponseType"/>
  </xs:complexContent>
</xs:complexType>
```

Figure 7. Reference Model Definition of Notify

5.2. <Notification> Element

The main content of the **<Notification>** element is the **<ItemData>** element, which contains the data the notification carries (e.g., current location, changed home address). In the case of a change notification, the same formats as in responses to queries for changed data are used.

The **<ItemData>** element may also contain some other type of data indicating what kind of an event has happened. The whole **<ItemData>** element might not be used at all as it is possible to subscribe to notifications to indicate that an event has happened (e.g., data has changed without having the data in a notification message). The `subscriptionID` indicates what data has changed. For privacy reasons, this is the recommended alternative in many cases.

In addition to the **<ItemData>** element(s), the **<Notification>** element has a number of XML attributes. The `subscriptionID` XML attributes identifies the subscription based on which a notification is sent. So one **<Notification>** element carries information only related to one **<Subscription>**. A **<Notify>** element may carry multiple **<Notification>** elements.

One **<ItemData>** element MUST NOT contain more data that address by one **<ResultQuery>** or **<RefItem>** of the subscription.

The `expires` XML attribute is used to indicate in a notification message the time when the subscription will expire. In an administrative notification, the `endReason` XML attribute can be used to indicate the reason for the end of the subscription. This might give some indication to a WSC that a WSP is having some problems or whether it makes sense or not for a WSC to try to make the subscription again. The `endReason` XML attribute is not used in normal notifications, only when administrative notifications are used to notify that a subscription has ended. Possible values for the `endReason` XML attribute include:

urn:liberty:subs:endreason:unspecified The real reason is unspecified.

urn:liberty:subs:endreason:wscnotacknowledging A WSP cancels the subscription as it has not received acknowledgments from a WSC to the notification messages.

urn:liberty:subs:endreason:resourcedeleted The resource has been deleted so there is no data available anymore.

urn:liberty:subs:endreason:expired The subscription has expired. Either a WSC did not renew it in time or a WSP changed the expiration time.

urn:liberty:subs:endreason:credentialsexpired The credentials given for sending notifications have expired, thus a WSP is not capable of sending any more notifications. This notification might have to be sent just before the credentials are about to expire. Otherwise, even this notification can not be sent.

A WSP must be careful not to compromise the privacy of a Principal when sending the reason codes for ending a subscription.

5.3. <NotifyResponse> Element

Notifications are acknowledged using the **<NotifyResponse>** element. It contains only the **<Status>** element. A service specification MUST specify whether notifications acknowledgments are used or not or whether it is an implementation- or deployment-specific decision.

5.4. Processing Rules for Notifications

The common processing rules specified in [LibertyDST], Section 3 "Message Interface," also MUST be followed.

1. A WSP MUST send a notification message to a WSC which has made a subscription when an event defined by the parameters of that subscription happens. When sending these normal notification message to a WSC, a WSP MUST use the information provided in the XML attribute `notifyToRef` element (i.e., endpoint, security mechanism, and credentials or tokens).

2. When a subscription becomes invalid or has been changed by a WSP somehow, a WSP MUST send a notification to indicate this if administrative notifications about subscriptions are used. When a WSP is sending a notification about expiration or change of a subscription, it MUST use the information provided in the XML attribute `adminNotifyToRef` (i.e., endpoint, security mechanism, and credentials or tokens). If the `adminNotifyToRef` XML attribute is not specified, the `notifyToRef` element is used instead.

3. If the receiving WSC can not successfully process one of the **<Notification>** elements inside one **<Notify>** element, it SHOULD process, normally, the rest of the **<Notification>** elements and try to achieve a partial success. A WSC MUST support multiple **<Notification>** elements inside one **<Notify>** element.

4. A **<Notification>** element inside a notification message MUST have a `subscriptionID` XML attribute to identify the subscription based on which the notification message is sent. If the `subscriptionID` XML attribute is missing, the processing of that **<Notification>** element MUST fail and the second level status code `MissingSubscriptionID` SHOULD be used, in addition to a top level status code, to indicate this. If a WSC does not recognize the value of a `subscriptionID` XML attribute, the processing of that **<Notification>** element MUST fail and the second level status code `InvalidSubscriptionID` SHOULD be used, in addition to a top level status code, to indicate this.

5. A **<Notification>** element inside a notification message MUST have the `expires` XML attribute, if subscription expiration is used. When a WSC receiving a notification knows that the `expires` XML attribute should have been used, but it is not, it SHOULD use the second level status code `MissingExpiration`. Irrespective of reporting the missing `expires`, the WSC MAY decide whether it considers this a failure or not.

6. One **<Notification>** element MUST NOT contain both the data subscribed and information about the change of a subscription. The only exception is the expiration time. If a WSP changes the expiration time, an administrative notification is sent, if used, but the new expiration time is also included in the normal notifications.

7. If a **<Notification>** element is supposed to contain data about a resource (i.e., the `includeData` XML attribute of a subscription has either the value `Yes` or `YesWithCommonAttributes`), the **<ItemData>** element MUST be used in a **<Notification>** element. The content of an **<ItemData>** element MUST be according to the parameters of the related subscription, especially **<ResultQuery>** or **<RefItem>**, and the related event which has caused this **<Notification>** element to be sent inside a notification message. In the case of a change notification, the same formatting rules for the content, as in the case of a query for changes, MUST be followed (see [LibertyDST], Section 4 "Querying Data"). A WSP MUST NOT include any data which the WSC is not allowed to get based on access rights and privacy policies defined by the resource owner. If an **<ItemData>** element should have been included in a **<Notification>** element, but it is missing, the processing of the **<Notification>** element MUST fail and the second level status code `MissingDataElement` SHOULD be used, in addition to the top level status code, to indicate this.

8. For change notification, a `changeFormat` XML attribute MUST be added for an **<ItemData>** element to indicate the format used to show the changes if a service specification has not mandated only one specific format to be used for this.

9. If the data inside an **<ItemData>** element is invalid, the processing of the **<Notification>** element MUST fail and the second level status code `InvalidData` SHOULD be used, in addition to the top level status code, to indicate this. A WSC MUST accept all the data which can be considered as possible normal extension if extensions are allowed for a service based on the service specification.

637 10. If a **<Notification>** element has the `endReason` XML attribute, the notification is expected to indicate the end
638 of the subscription and all other content of the **<Notification>** element. The exception to this is that the
639 `subscriptionID` XML attribute MUST be ignored unless some service-specific extensions needed in these
640 kinds of cases have been specified. The `endReason` XML attribute MUST have a value specified in this document
641 or valid service or implementation-specific value. A WSP MUST be careful not to use any value which might
642 compromise the privacy of a Principal.

643 11. A WSP SHOULD resend a notification for which it does not get an acknowledgment in reasonable time if
644 acknowledgments are used. If a WSP does not get acknowledgments at all within its time and other limits, it
645 MAY cancel the related subscription.

6. Subscription and Notification Examples

6.1. Piggy-Backing a Subscription to Query

Consider a subscription to data that is queried.

```
<Query>
  <QueryItem itemID="djkfgjkdf">
    <Select>/hp:HP/hp:AddressCard</Select>
  </QueryItem>
  <Subscription
     includeData="Yes"
     subscriptionID="tr578k-kydg4b"
     notifyToRef="#123">
    <RefItem itemIDRef="djkfgjkdf"/>
  </Subscription>
</Query>
```

Here we see `itemIDRef` referencing the **<QueryItem>** to define the data to be subscribed. The subscriber also allocates a `subscriptionID` and provides the end point to contact by way of the `notifyToRef` XML attribute that references an endpoint in the SOAP headers (not shown).

This subscription could later generate following notification.

```
<Notify>
  <Notification subscriptionID="tr578k-kydg4b">
    <ItemData>
      <hp:AddressCard id="9812">
        <hp:AddressType>urn:liberty:id-sis-hp:addrType:home</hp:AddressType>
        <hp:Address>
          <hp:C>us</hp:C>
        </hp:Address>
      </hp:AddressCard>
    </ItemData>
  </Notification>
</Notify>
```

The salient point to notice is that the **<Notification>** correlates to the subscription using the `subscriptionID` XML attribute.

6.2. Creating Subscription Object

Consider:

```
<Create>
  <CreateItem objectType="_Subscription" itemID="1">
    <NewData>
      <Subscription
         subscriptionID="subs123"
         notifyToRef="#1"
         includeData="1">
        <ResultQuery objectType="entry">
          <Select attributes="HELLO"/>
        </ResultQuery>
        <RefItem itemIDRef="1"/>
      </Subscription>
    </NewData>
  </CreateItem>
</Create>
```

699
700

701 The above example illustrates:

702 a. Creating a subscription by explicit creation of an object of type "_Subscription,"

703 b. Defining notification data using **<ResultQuery>**,

704 c. Creating a subscription to the data of **<CreateItem>** by referencing it using **<RefItem>**, and

705 d. Subscribing to the changes to the subscription itself.

7. Checklist for Service Specifications

1. Provide schema for <**Notify**>, <**NotifyResponse**>, and <**Subscription**> elements. If these are named differently, indicate the correspondence to the standard naming.

2. If the service supports subscriptions, it will need to handle the object type "_Subscription." If the `AppDataType` is defined using XML schema, this schema needs to make allowance (e.g., by using the <**xs:choice**> construct) for <**Subscription**> elements by referencing the DST schema.

   If the service adopts the default definition of `AppDataType`, which uses the *mixed* content model, then the default is that all string data belongs to the service-specified object types while any <**Subscription**> containers belong to object type "_Subscription." Further, an element of type `AppDataType` may only contain objects of one type.

3. Describe how `SelectType` applies to subscriptions.

   In particular, if the service supports subscriptions, it MUST provide a way to specify XPath expressions for querying them. The XPath expressions MAY be restricted to the subset described in Section 4.6. This MAY be specified by stating that "default restriction on XPaths for subscriptions applies."

4. Describe the `TriggerType` or state that is not used.

5. Describe the `AggregationType` or state that is not used.

6. Extension support.

   a. If `TriggerType` or `AggregationType` is designated as unused by the service specification, then it MAY be used for extension, provided that the extension data is

      a. In URI format and use an assigned domain name as a component of the URI to ensure that extensions do no collide with each other.

      b. A namespace-qualified XML document.

7. Statement of how subscriptions can be established and manipulated.

   a. Support CRUD manipulation of subscriptions as `objectType` "_Subscription."

   b. Support subscribing in <**Query**>.

   c. Support multiple <**Subscription**> elements in <**Query**>.

   d. Support subscribing in <**Create**>.

   e. Support multiple <**Subscription**> elements in <**Create**>.

   f. Support subscribing in <**Modify**>.

   g. Support multiple <**Subscription**> elements in <**Modify**>.

8. Start of a subscription. Usually, a subscription is valid after it has been created, but, if supported, a WSC may request that a subscription is valid only after a specific time using the `starts` XML attribute. It MUST be specified here whether the `starts` XML attribute is supported or not.

9. Subscription expiration. Usually, subscriptions expire after a certain time, but a service specification may also specify, for example, that subscription expiration is not used and WSCs must cancel subscriptions after they are not needed. It MUST be specified here whether subscriptions expire or not (e.g., Subscription expiration MUST be used).

10. Support `expires==starts`. Is specifying the same time for both the `starts` and `expires` XML attributes to request one notification message at a specified time (e.g., same value MAY be used both for the `starts` and the `expires` XML attribute) allowed?

11. Support querying existing subscriptions. Some services or implementations may or may not support querying existing subscriptions. This should be stated here (e.g., MUST NOT be supported).

12. Support acknoledging notifications. Some services or implementations may or may not support acknowledging notifications using **<NotifyResponse>**. This should be stated here (e.g., Notifications MUST BE acknowledged).

8. Schemata

8.1. Schema for DST Reference Model with Subscriptions and Notifications

The formal schema for the reference model follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   targetNamespace="urn:liberty:ssos:2006-08:ref"
   xmlns:subsref="urn:liberty:ssos:2006-08:ref"
   xmlns:dst="urn:liberty:dst:2006-08"
   xmlns:subs="urn:liberty:ssos:2006-08"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
  <xs:import namespace="urn:liberty:dst:2006-08"
     schemaLocation="liberty-idwsf-dst-v2.1.xsd"/>
  <xs:import namespace="urn:liberty:ssos:2006-08"
     schemaLocation="liberty-idwsf-subs-v1.0.xsd"/>
  <xs:import namespace="urn:liberty:util:2006-08"
     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
<!--sec(methods)-->
  <xs:element name="Create" type="subsref:CreateType"/>
  <xs:element name="CreateResponse" type="subsref:CreateResponseType"/>
  <xs:element name="Query" type="subsref:QueryType"/>
  <xs:element name="QueryResponse" type="subsref:QueryResponseType"/>
  <xs:element name="Modify" type="subsref:ModifyType"/>
  <xs:element name="ModifyResponse" type="subsref:ModifyResponseType"/>
  <xs:element name="Delete" type="subsref:DeleteType"/>
  <xs:element name="DeleteResponse" type="subsref:DeleteResponseType"/>
<!--endsec(methods)-->
<!--sec(notifymethods)-->
  <xs:element name="Notify" type="subsref:NotifyType"/>
  <xs:element name="NotifyResponse" type="subsref:NotifyResponseType"/>
<!--endsec(notifymethods)-->
<!--sec(redefs)-->
  <xs:complexType name="SelectType">
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="TestOpType">
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="SortType">
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="TriggerType">
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="AggregationType">
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType mixed="1" name="AppDataType">
    <xs:sequence>
```

```
      <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
<!--endsec(redefs)-->
<!--sec(create)-->
  <xs:complexType name="CreateType">
    <xs:complexContent>
      <xs:extension base="dst:RequestType">
        <xs:sequence>
          <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="subsref:CreateItem" minOccurs="1" maxOccurs="unbounded"/>
          <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(create)-->
<!--sec(createaux)-->
  <xs:element name="CreateItem" type="subsref:CreateItemType"/>
  <xs:complexType name="CreateItemType">
    <xs:sequence>
      <xs:element ref="subsref:NewData" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attributeGroup ref="dst:CreateItemAttributeGroup"/>
  </xs:complexType>
  <xs:element name="NewData" type="subsref:AppDataType"/>
  <xs:complexType name="CreateResponseType">
    <xs:complexContent>
      <xs:extension base="subsref:DataResponseType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="DataResponseType">
    <xs:complexContent>
      <xs:extension base="dst:DataResponseBaseType">
        <xs:sequence>
          <xs:element ref="subsref:ItemData" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(createaux)-->
<!--sec(query)-->
  <xs:complexType name="QueryType">
    <xs:complexContent>
      <xs:extension base="dst:RequestType">
        <xs:sequence>
          <xs:element ref="subsref:TestItem" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="subsref:QueryItem" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(query)-->
<!--sec(queryaux)-->
  <xs:element name="TestItem" type="subsref:TestItemType"/>
  <xs:complexType name="TestItemType">
    <xs:complexContent>
      <xs:extension base="dst:TestItemBaseType">
        <xs:sequence>
          <xs:element name="TestOp" minOccurs="0" maxOccurs="1" type="subsref:TestOpType"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="QueryItem" type="subsref:QueryItemType"/>
  <xs:complexType name="QueryItemType">
```

```
    <xs:complexContent>
      <xs:extension base="subsref:ResultQueryType">
        <xs:attributeGroup ref="dst:PaginationAttributeGroup"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(queryaux)-->
<!--sec(queryresp)-->
  <xs:complexType name="QueryResponseType">
    <xs:complexContent>
      <xs:extension base="dst:DataResponseBaseTyp e">
        <xs:sequence>
          <xs:element ref="lu:TestResult" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="subsref:Data" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="Data" type="subsref:DataType"/>
  <xs:complexType name="DataType">
    <xs:complexContent>
      <xs:extension base="subsref:ItemDataType">
        <xs:attributeGroup ref="dst:PaginationResponseAttr ibuteGroup"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(queryresp)-->
<!--sec(mod)-->
  <xs:complexType name="ModifyType">
    <xs:complexContent>
      <xs:extension base="dst:RequestType">
        <xs:sequence>
          <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="subsref:ModifyItem" minOccurs="1" maxOccurs="unbounded"/>
          <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(mod)-->
<!--sec(modaux)-->
  <xs:element name="ModifyItem" type="subsref:ModifyItemType"/>
  <xs:complexType name="ModifyItemType">
    <xs:sequence>
      <xs:element ref="subsref:Select" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="subsref:NewData" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attributeGroup ref="dst:ModifyItemAttributeGroup"/>
  </xs:complexType>
  <xs:complexType name="ModifyResponseType">
    <xs:complexContent>
      <xs:extension base="subsref:DataResponseType"/>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(modaux)-->
<!--sec(del)-->
  <xs:complexType name="DeleteType">
    <xs:complexContent>
      <xs:extension base="dst:RequestType">
        <xs:sequence>
          <xs:element ref="subsref:DeleteItem" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="DeleteItem" type="subsref:DeleteItemType"/>
  <xs:complexType name="DeleteItemType">
```

```
    <xs:complexContent>
     <xs:extension base="dst:DeleteItemBaseType">
      <xs:sequence>
       <xs:element ref="subsref:Select" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="DeleteResponseType">
    <xs:complexContent>
     <xs:extension base="lu:ResponseType"/>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(del)-->
<!--sec(resqry)-->
  <xs:element name="Select" type="subsref:SelectType"/>
  <xs:element name="ResultQuery" type="subsref:ResultQueryType"/>
  <xs:complexType name="ResultQueryType">
    <xs:complexContent>
     <xs:extension base="dst:ResultQueryBaseType">
      <xs:sequence>
       <xs:element ref="subsref:Select" minOccurs="0" maxOccurs="1"/>
       <xs:element name="Sort" minOccurs="0" maxOccurs="1" type="subsref:SortType"/>
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="ItemData" type="subsref:ItemDataType"/>
  <xs:complexType name="ItemDataType">
    <xs:complexContent>
     <xs:extension base="subsref:AppDataType">
      <xs:attributeGroup ref="dst:ItemDataAttributeGroup"/>
     </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(resqry)-->
<!--sec(subscr)-->
  <xs:element name="Subscription" type="subsref:SubscriptionType"/>
  <xs:complexType name="SubscriptionType">
    <xs:complexContent>
     <xs:extension base="subs:SubscriptionType">
      <xs:sequence>
       <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
       <xs:element name="Aggregation" minOccurs="0" maxOccurs="1" type="subsref:AggregationType"/>
       <xs:element name="Trigger" minOccurs="0" maxOccurs="1" type="subsref:TriggerType"/>
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(subscr)-->
<!--sec(notif)-->
  <xs:complexType name="NotifyType">
    <xs:complexContent>
     <xs:extension base="dst:RequestType">
      <xs:sequence>
       <xs:element ref="subsref:Notification" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
     </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="Notification" type="subsref:NotificationType"/>
  <xs:complexType name="NotificationType">
    <xs:complexContent>
     <xs:extension base="subs:NotificationType">
      <xs:sequence>
       <xs:element ref="subsref:ItemData" minOccurs="0" maxOccurs="unbounded"/>
```

```
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 <xs:complexType name="NotifyResponseType">
  <xs:complexContent>
   <xs:extension base="subs:NotifyResponseType"/>
  </xs:complexContent>
 </xs:complexType>
<!--endsec(notif)-->
</xs:schema>
```

8.2. Subscriptions Utility Schema

The formal utility schema follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   targetNamespace="urn:liberty:ssos:2006-08"
   xmlns:subs="urn:liberty:ssos:2006-08"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
 <xs:import namespace="urn:liberty:util:2006-08"
    schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
<!--sec(subscr)-->
 <xs:complexType name="SubscriptionType">
  <xs:sequence>
   <xs:element ref="subs:RefItem" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="lu:Extension" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="subscriptionID" use="required" type="lu:IDType"/>
  <xs:attribute name="notifyToRef" use="required" type="xs:anyURI"/>
  <xs:attribute name="adminNotifyToRef" use="optional" type="xs:anyURI"/>
  <xs:attribute name="starts" use="optional" type="xs:dateTime"/>
  <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
  <xs:attribute name="id" use="optional" type="xs:ID"/>
  <xs:attribute name="includeData" use="optional">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="Yes"/>
     <xs:enumeration value="No"/>
     <xs:enumeration value="YesWithCommonAttributes"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
 <xs:element name="RefItem" type="subs:RefItemType"/>
 <xs:complexType name="RefItemType">
  <xs:attribute name="subscriptionID" use="optional" type="lu:IDType"/>
  <xs:attribute ref="lu:itemIDRef" use="required"/>
 </xs:complexType>
<!--endsec(subscr)-->
<!--sec(notif)-->
 <xs:attributeGroup name="NotifyAttributeGroup">
  <xs:attribute name="timeStamp" use="optional" type="xs:dateTime"/>
 </xs:attributeGroup>
 <xs:complexType name="NotificationType">
  <xs:sequence>
   <xs:element ref="lu:TestResult" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" use="optional" type="xs:ID"/>
```

```
    <xs:attribute name="subscriptionID" use="required" type="lu:IDType"/>
    <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
    <xs:attribute name="endReason" use="optional" type="xs:anyURI"/>
  </xs:complexType>
  <xs:complexType name="NotifyResponseType">
    <xs:complexContent>
      <xs:extension base="lu:ResponseType"/>
    </xs:complexContent>
  </xs:complexType>
<!--endsec(notif)-->
</xs:schema>
```

References

Normative

[LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14 December, 2004). *http://www.projectliberty.org/specs*

[LibertyDST] Kellomäki, Sampo, Kainulainen, Jukka, eds. "Liberty ID-WSF Data Services Template," Version 2.1, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyDisco] Hodges, Jeff, Cahill, Conor, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty ID-WSF SOAP Binding Specification," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version v2.0, Liberty Alliance Project (30 July, 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*

[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/*

[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*

[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*

[XPATH] Clark , J., DeRose , S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0 ," Recommendation, W3C *http://www.w3.org/TR/xpath [August 2003].*

Informative

[LibertyPeopleService] Koga, Yuzo, Madsen, Paul, eds. "Liberty ID-WSF People Service Specification," Version 1.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*