



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-subsv1.0.pdf

1 **Notice**

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

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

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

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

34 Contents

35	1. Introduction	4
36	1.1. Notation	4
37	1.2. Liberty Considerations	4
38	1.3. Namespaces	4
39	1.4. Applying Subscriptions to DST-Based Services	4
40	2. General Rules Regarding Subscriptions and Notifications	6
41	2.1. Second Level <Status> Codes for Subscriptions	6
42	2.2. Discovery Option Keywords for Indicating Lack of Subscription Support	6
43	2.3. CRUD Manipulation Using Object Type "_Subscription"	6
44	2.4. No itemIDRef for Subscription-Related <ItemData>	6
45	3. Piggy-Backing Subscriptions to DST Operations	7
46	3.1. <Query> with <Subscription>	7
47	3.2. <Create> with <Subscription>	7
48	3.3. <Modify> with <Subscription>	7
49	4. Subscriptions	9
50	4.1. <Subscription> element	9
51	4.2. Selecting Data to which a Subscription Applies	10
52	4.3. Providing Information for Sending Notifications	10
53	4.4. Expiration of Subscription	11
54	4.5. Common Processing Rules for Subscriptions	11
55	4.5.1. General Processing Rules for Subscriptions	11
56	4.5.2. Processing Rules for Data to which the Subscription Applies	12
57	4.5.3. Processing Rules for <Aggregation> and <Trigger>	12
58	4.5.4. Processing Rules for First Notification and Expiry of Subscription	13
59	4.5.5. Processing Rules When the Access and Privacy Policies Forbid Subscription	13
60	4.6. selectType for Subscription Objects	14
61	4.7. Support for <Subscription> Conditioned by <TestItem>	14
62	5. Notifications	16
63	5.1. <Notify> Element	16
64	5.2. <Notification> Element	16
65	5.3. <NotifyResponse> Element	17
66	5.4. Processing Rules for Notifications	17
67	6. Subscription and Notification Examples	20
68	6.1. Piggy-Backing a Subscription to Query	20
69	6.2. Creating Subscription Object	20
70	7. Checklist for Service Specifications	22
71	8. Schemata	24
72	8.1. Schema for DST Reference Model with Subscriptions and Notifications	24
73	8.2. Subscriptions Utility Schema	28
74	References	30

75 1. Introduction

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

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

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

86 1.1. Notation

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

90 Definitions for Liberty-specific terms can be found in [[LibertyGlossary](#)].

91 1.2. Liberty Considerations

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

96 1.3. Namespaces

97 The namespaces described in Table 1 are used.

98 **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 [Liberty-Disco].
sb:	urn:liberty:sb:2006-08	Liberty ID-WSF SOAP Binding Extension [LibertySOAPBinding].

99 **1.4. Applying Subscriptions to DST-Based Services**

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

103 Specifically, the service is expected to provide definitions for

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

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

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

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

114 2. General Rules Regarding Subscriptions and Notifications

115 2.1. Second Level <Status> Codes for Subscriptions

116 The following second level status codes are defined for subscriptions:

```
117 EmbeddedSubscriptionsNotSupported
118 InvalidSubscriptionID
119 MissingSubscriptionID
120
121
```

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

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

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

133 2.2. Discovery Option Keywords for Indicating Lack of Subscription 134 Support

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

```
137 urn:liberty:subs:noSubscribe
138 urn:liberty:subs:noQuerySubscriptions
139
140
```

141 2.3. CRUD Manipulation Using Object Type "_Subscription"

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

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

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

150 2.4. No `itemIDRef` for Subscription-Related `<ItemData>`

151 The `<ItemData>` elements returning changed expiration times for subscriptions created based on the request mes-
152 sage MUST NOT contain any `itemIDRef` XML attribute. They contain `<Subscription>` elements, which carry
153 `subscriptionID` XML attributes (see [Section 4](#)).

154 3. Piggy-Backing Subscriptions to DST Operations

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

156 3.1. <Query> with <Subscription>

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

```

163 <xs:complexType name="QueryType">
164   <xs:complexContent>
165     <xs:extension base="dst:RequestType">
166       <xs:sequence>
167         <xs:element ref="subsref:TestItem" minOccurs="0" maxOccurs="unbounded"/>
168         <xs:element ref="subsref:QueryItem" minOccurs="0" maxOccurs="unbounded"/>
169         <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
170       </xs:sequence>
171     </xs:extension>
172   </xs:complexContent>
173 </xs:complexType>
  
```

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

175 3.2. <Create> with <Subscription>

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

```

178 <xs:complexType name="CreateType">
179   <xs:complexContent>
180     <xs:extension base="dst:RequestType">
181       <xs:sequence>
182         <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded"/>
183         <xs:element ref="subsref:CreateItem" minOccurs="1" maxOccurs="unbounded"/>
184         <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
185       </xs:sequence>
186     </xs:extension>
187   </xs:complexContent>
188 </xs:complexType>
  
```

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

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

194 3.3. <Modify> with <Subscription>

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

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

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

```
202 <xs:complexType name="ModifyType">
203   <xs:complexContent>
204     <xs:extension base="dst:RequestType">
205       <xs:sequence>
206         <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded" />
207         <xs:element ref="subsref:ModifyItem" minOccurs="1" maxOccurs="unbounded" />
208         <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded" />
209       </xs:sequence>
210     </xs:extension>
211   </xs:complexContent>
212 </xs:complexType>
```

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

214 4. Subscriptions

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

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

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

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

233 Notifications are carried inside `<Notify>` elements. The notifications are specified in [Section 5](#).

234 4.1. `<Subscription>` element

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

```

237 <xs:complexType name="SubscriptionType">
238   <xs:sequence>
239     <xs:element ref="subs:RefItem" minOccurs="0" maxOccurs="unbounded" />
240     <xs:element ref="lu:Extension" minOccurs="0" maxOccurs="unbounded" />
241   </xs:sequence>
242   <xs:attribute name="subscriptionID" use="required" type="lu:IDType" />
243   <xs:attribute name="notifyToRef" use="required" type="xs:anyURI" />
244   <xs:attribute name="adminNotifyToRef" use="optional" type="xs:anyURI" />
245   <xs:attribute name="starts" use="optional" type="xs:dateTime" />
246   <xs:attribute name="expires" use="optional" type="xs:dateTime" />
247   <xs:attribute name="id" use="optional" type="xs:ID" />
248   <xs:attribute name="includeData" use="optional" />
249   <xs:simpleType>
250     <xs:restriction base="xs:string">
251       <xs:enumeration value="Yes" />
252       <xs:enumeration value="No" />
253       <xs:enumeration value="YesWithCommonAttributes" />
254     </xs:restriction>
255   </xs:simpleType>
256 </xs:attribute>
257 </xs:complexType>
258 <xs:element name="RefItem" type="subs:RefItemType" />
259 <xs:complexType name="RefItemType">
260   <xs:attribute name="subscriptionID" use="optional" type="lu:IDType" />
261   <xs:attribute ref="lu:itemIDRef" use="required" />
262 </xs:complexType>
    
```

263 **Figure 4. Utility Schema for Subscription**

```
264 <xs:element name="Subscription" type="subsref:SubscriptionType"/>
265 <xs:complexType name="SubscriptionType">
266   <xs:complexContent>
267     <xs:extension base="subs:SubscriptionType">
268       <xs:sequence>
269         <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
270         <xs:element name="Aggregation" minOccurs="0" maxOccurs="1" type="subsref:AggregationType"/>
271         <xs:element name="Trigger" minOccurs="0" maxOccurs="1" type="subsref:TriggerType"/>
272       </xs:sequence>
273     </xs:extension>
274   </xs:complexContent>
275 </xs:complexType>
```

276 **Figure 5. Reference Model Definition of Subscription**

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

282 4.2. Selecting Data to which a Subscription Applies

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

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

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

296 4.3. Providing Information for Sending Notifications

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

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

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

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

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

316 4.4. Expiration of Subscription

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

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

323 4.5. Common Processing Rules for Subscriptions

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

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

327 4.5.1. General Processing Rules for Subscriptions

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

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

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

- 355 4. The values of the `subscriptionID` XML attributes are WSC-specific. When a new subscription is created, it
356 MUST use a `subscriptionID` different from any other subscription the same WSC has at the same WSP. If a
357 WSC tries to create a new subscription which has a conflicting `subscriptionID` value, a WSP MUST reject
358 that and it SHOULD use the second level status code `InvalidSubscriptionID`.
- 359 5. An implementation MAY decompose a composite subscription object into unit subscriptions. For example, if
360 a subscription object has multiple `<RefItem>` elements with different `subscriptionID` XML attributes, this
361 is interpreted to create multiple logical subscriptions. An implementation may, indeed, choose to handle them
362 as separate subscriptions. While an implementation MUST support the creation of composite subscriptions, it
363 NEED NOT support composite subscriptions on `<Query>`, `<Modify>`, and `<Delete>` interfaces involving objects
364 of type "`_Subscription`."

365 4.5.2. Processing Rules for Data to which the Subscription Applies

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

- 367 1. When `<Subscription>` elements contain a `<ResultQuery>` element, a WSP MUST process its content in a similar
368 fashion as it processes the same parameters in the case of a normal query, taking into account that no data is
369 returned immediately. A WSP MUST support the requested `objectType` and `<Select>`. If a WSP does not
370 support sorting and it is requested by a WSC, a WSP SHOULD still accept the subscriptions and return data
371 unsorted in notifications. The `changedSince` XML attribute MUST be ignored, if present. When notifications
372 are expected to contain only the changed data, a WSC MAY use `<ChangeFormat>` to indicate formats it supports.
373 Note that with subscriptions, the `<ChangeFormat>` is used without having the `changedSince` XML attribute
374 (required in regular queries). The `predefined` XML attribute can be used instead of other parameters. See
375 [[LibertyDST](#)], Section 3.7 "Selection" and Section 4.4 "Processing Rules for Queries" for more details and proper
376 status codes.
- 377 2. When a `<RefItem>` element is included in a subscription, it MUST contain an `itemIDRef` XML attribute. The
378 value of this XML attribute MUST be the same as the value of an `itemID` XML attribute of a `<QueryItem>`,
379 a `<CreateItem>`, or a `<ModifyItem>`, depending on the message. This creates a subscription to all of the data
380 manipulated in the referenced element.
- 381 3. If the value of the `itemIDRef` XML attribute does not match to any relevant `itemID`, the subscription MUST
382 NOT be accepted and the second level status code `InvalidItemIDRef` SHOULD be used to indicate the reason.
- 383 4. If a `<RefItem>` element contains a `subscriptionID` XML attribute and it has a different value than the
384 `subscriptionID` XML attribute of the `<Subscription>` element, the `<RefItem>` element defines a new sub-
385 scription which inherits other parameters, except `<ResultQuery>` elements, `subscriptionID` XML attribute,
386 and possible other `<RefItem>` elements from the `<Subscription>` element in which the `<RefItem>` element is
387 contained. Each `<RefItem>` that has a `subscriptionID` XML attribute creates a new independent subscription.
388 If multiple `<RefItem>` elements have the same value of the `subscriptionID`, they all form one subscription
389 together and that subscription has multiple sets of selection parameters. If data selected by any of the sets is
390 changed, a notification is sent.
- 391 5. A `<Subscription>` element may contain any number of `<ResultQuery>`, `<RefItem>`, and `<Trigger>` elements.
392 If none of the elements `<ResultQuery>`, `<RefItem>` or `<Trigger>` are present, the processing of the `<Subscrip-`
393 `tion>` element MUST fail unless the service specification has defined, what this kind of a case means, e.g., some
394 default values are defined for parameters and those are used or a WSC subscribes to the whole resource. When
395 the processing of a `<Subscription>` element fails due to not having `<ResultQuery>`, `<RefItem>` or `<Triggers>`
396 present, the second level status code `MissingSelect` SHOULD be used to indicate this.

397 4.5.3. Processing Rules for `<Aggregation>` and `<Trigger>`

- 398 1. A WSP MUST follow the processing rules defined in the service specification for the elements **<Aggregation>**
399 and **<Trigger>**. If the use of these elements is not specified for the service or specified, but not supported
400 by a WSP, and either of both of them are included in a **<Subscription>** element in a **<Subscribe>** request,
401 the processing of the **<Subscription>** MUST fail and a second level status code SHOULD be used, either
402 `AggregationNotSupported` or `TriggerNotSupported`, to indicate this.
- 403 2. If a WSP does support aggregation, but not the type of **<Aggregation>** a WSC requests, the processing of
404 the **<Subscription>** MUST fail and the second level status code `RequestedAggregationNotSupported`
405 SHOULD be used, in addition to the top level status code, to indicate this. Similarly, if a WSP does support
406 triggers, but not the type of a **<Trigger>** a WSC requests, the processing of the **<Subscription>** MUST fail and
407 the second level status code `RequestedTriggerNotSupported` SHOULD be used, in addition to the top level
408 status code, to indicate this.

409 4.5.4. Processing Rules for First Notification and Expiry of Subscription

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

- 411 1. If a **<Subscription>** element contains a `starts` XML attribute, subscription MUST be valid *only* after the time
412 defined. If the `starts` XML attribute is omitted, the subscription MUST be valid immediately after processing
413 the request. Also, if the time specified by the `starts` XML attribute is in the past, then that subscription, if
414 accepted by a WSP, MUST be valid immediately after processing the request.
- 415 2. The time specified by the `expires` XML attribute MUST be the same time or a later time than the time specified
416 by the `starts` XML attribute in the same **<Subscription>** element. It also MUST be later than the current time.
417 If either of the checks is not passed, then the processing of the **<Subscription>** MUST fail and the second level
418 status code `InvalidExpires` SHOULD be used, in addition to the top level status code, to indicate this.
- 419 3. A WSP MAY change the time when a subscription expires from the expiration time requested by a WSC with
420 the `expires` XML attribute. A WSP MAY shorten the expiration time, but it MUST NOT make the expiration
421 time longer. If no `expires` XML attribute is included in a **<Subscription>** element in a request from a WSC, a
422 WSP MUST decide the expiration time for the subscription, if expiration times are required either by the service
423 specification or the WSP. A WSP MUST return the expiration time in the response message if it is changed
424 compared to what a WSC requested. This information is returned by returning a **<Subscription>** element with
425 XML attributes `subscriptionID` and `expires` inside a **<Data>** element in the case of a **<QueryResponse>**
426 and inside a **<ItemData>** in the case of a **<CreateResponse>** and **<ModifyResponse>**. That **<Data>** or
427 **<ItemData>** element MUST NOT contain any other data than **<Subscription>** elements created based on one
428 **<Subscription>** element or, when a normal data object creation method has been used, **<Subscription>** elements
429 created with one **<CreateItem>** element. The **<Data>** or **<ItemData>** element SHOULD NOT contain any
430 `itemIDRef` XML attributes. The matching is done based on the `subscriptionID` XML attributes carried
431 inside **<Subscription>** elements.
- 432 4. If a WSC wants to renew an existing subscription before it has ended, it MUST modify that subscription and give
433 a new value for the `expires` XML attribute of that subscription. A WSP MAY modify the new value in the
434 same way as it MAY modify the proposed value for a new subscription.
- 435 5. There is one special case when using subscriptions expirations. When the `starts` and `expires` XML attributes
436 have exactly the same value, the meaning is that a notification MUST be sent exactly at that time whether some
437 event (e.g., data change) has happened or not. A WSC wants to get current values of the data (e.g., location)
438 exactly at that time, even if the values have stayed the same for a long time (e.g., a Principal has not moved).

439 4.5.5. Processing Rules When the Access and Privacy Policies Forbid 440 Subscription

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

- 443 1. When a WSP processes a **<Subscription>** element, it **MUST** check whether the resource owner (the Principal,
444 for example) has given consent to return the requested data and the fact that an event or data change has happened
445 in notification messages. To be able to check WSC-specific access rights, the WSP **MUST** authenticate the
446 WSC (see [LibertySecMech]). The WSP **MUST** also check that any usage directive given in the request is
447 acceptable based on the usage directives defined by the resource owner (see [LibertySOAPBinding]). If either
448 check fails, the WSP **MUST NOT** accept the subscription and the processing of that **<Subscription>** **MUST**
449 fail. The WSP **MAY** try to get consent from the Principal while processing the request, perhaps by using an
450 interaction service (see [LibertyInteract]). A WSP might check the access rights and policies in usage directives
451 at a higher level, before getting to DST processing and **MAY**, in this case, just return an ID-* Fault Message
452 (see [LibertySOAPBinding]) without processing the *RequestElement* element at all if the requesting WSC is not
453 allowed to access the data in question.
- 454 2. Note that there can be consent for subscribing to some data element but not its XML attributes. A Principal might
455 not want to release the *modifier* XML attribute if she does not want to reveal information about which services
456 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
457 subscription, but it **MAY** also reject it. If a subscription is accepted, the data for which there is no consent from
458 the Principal **MUST** be handled as if there were no data. Also that data, or the fact that the data has changed,
459 **MUST NOT** be included in the notification messages sent later on.
- 460 3. If a WSC has made a subscription and included the usage directive it has promised to obey, then later wants to
461 change the usage directive, it **MUST** cancel the subscription and make a new subscription with the new value for
462 the usage directive.

463 4.6. selectType for Subscription Objects

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

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

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

474 A WSP **MAY** also support:

```
475 /ns:Subscription[@ns:subscriptionID="xx"]/ns:notifyToRef  
476 /ns:Subscription[@ns:subscriptionID="xx"]/ns:adminNotifyToRef  
477  
478
```

479 to make it possible to change endpoints and related information.

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

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

484 4.7. Support for <Subscription> Conditioned by <TestItem>

485 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,
486 the WSC can ask to be notified when the result becomes false and vice versa.

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

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

491 `urn:liberty:subs:contingentSubscription`
492
493

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

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

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

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

504 5. Notifications

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

508 5.1. <Notify> Element

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

```

511 <xs:attributeGroup name="NotifyAttributeGroup">
512   <xs:attribute name="timeStamp" use="optional" type="xs:dateTime"/>
513 </xs:attributeGroup>
514 <xs:complexType name="NotificationType">
515   <xs:sequence>
516     <xs:element ref="lu:TestResult" minOccurs="0" maxOccurs="unbounded"/>
517   </xs:sequence>
518   <xs:attribute name="id" use="optional" type="xs:ID"/>
519   <xs:attribute name="subscriptionID" use="required" type="lu:IDType"/>
520   <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
521   <xs:attribute name="endReason" use="optional" type="xs:anyURI"/>
522 </xs:complexType>
523 <xs:complexType name="NotifyResponseType">
524   <xs:complexContent>
525     <xs:extension base="lu:ResponseType"/>
526   </xs:complexContent>
527 </xs:complexType>
    
```

528 **Figure 6. Utility Schema for Notify**

```

529 <xs:complexType name="NotifyType">
530   <xs:complexContent>
531     <xs:extension base="dst:RequestType">
532       <xs:sequence>
533         <xs:element ref="subsref:Notification" minOccurs="0" maxOccurs="unbounded"/>
534       </xs:sequence>
535       <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
536     </xs:extension>
537   </xs:complexContent>
538 </xs:complexType>
539 <xs:element name="Notification" type="subsref:NotificationType"/>
540 <xs:complexType name="NotificationType">
541   <xs:complexContent>
542     <xs:extension base="subs:NotificationType">
543       <xs:sequence>
544         <xs:element ref="subsref:ItemData" minOccurs="0" maxOccurs="unbounded"/>
545       </xs:sequence>
546     </xs:extension>
547   </xs:complexContent>
548 </xs:complexType>
549 <xs:complexType name="NotifyResponseType">
550   <xs:complexContent>
551     <xs:extension base="subs:NotifyResponseType"/>
552   </xs:complexContent>
553 </xs:complexType>
    
```

554 **Figure 7. Reference Model Definition of Notify**

555 5.2. <Notification> Element

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

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

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

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

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

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

576 `urn:liberty:subs:endreason:wscnotacknowledging` A WSP cancels the subscription as it has not received acknowl-
577 edgments from a WSC to the notification messages.

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

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

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

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

587 **5.3. <NotifyResponse> Element**

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

591 **5.4. Processing Rules for Notifications**

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

- 593 1. A WSP MUST send a notification message to a WSC which has made a subscription when an event defined
594 by the parameters of that subscription happens. When sending these normal notification message to a WSC, a
595 WSP MUST use the information provided in the XML attribute `notifyToRef` element (i.e., endpoint, security
596 mechanism, and credentials or tokens).
- 597 2. When a subscription becomes invalid or has been changed by a WSP somehow, a WSP MUST send a notification
598 to indicate this if administrative notifications about subscriptions are used. When a WSP is sending a notification
599 about expiration or change of a subscription, it MUST use the information provided in the XML attribute
600 `adminNotifyToRef` (i.e., endpoint, security mechanism, and credentials or tokens). If the `adminNotifyToRef`
601 XML attribute is not specified, the `notifyToRef` element is used instead.
- 602 3. If the receiving WSC can not successfully process one of the `<Notification>` elements inside one `<Notify>`
603 element, it SHOULD process, normally, the rest of the `<Notification>` elements and try to achieve a partial
604 success. A WSC MUST support multiple `<Notification>` elements inside one `<Notify>` element.
- 605 4. A `<Notification>` element inside a notification message MUST have a `subscriptionID` XML attribute to
606 identify the subscription based on which the notification message is sent. If the `subscriptionID` XML
607 attribute is missing, the processing of that `<Notification>` element MUST fail and the second level status code
608 `MissingSubscriptionID` SHOULD be used, in addition to a top level status code, to indicate this. If a
609 WSC does not recognize the value of a `subscriptionID` XML attribute, the processing of that `<Notification>`
610 element MUST fail and the second level status code `InvalidSubscriptionID` SHOULD be used, in addition
611 to a top level status code, to indicate this.
- 612 5. A `<Notification>` element inside a notification message MUST have the `expires` XML attribute, if subscription
613 expiration is used. When a WSC receiving a notification knows that the `expires` XML attribute should have
614 been used, but it is not, it SHOULD use the second level status code `MissingExpiration`. Irrespective of
615 reporting the missing `expires`, the WSC MAY decide whether it considers this a failure or not.
- 616 6. One `<Notification>` element MUST NOT contain both the data subscribed and information about the change of a
617 subscription. The only exception is the expiration time. If a WSP changes the expiration time, an administrative
618 notification is sent, if used, but the new expiration time is also included in the normal notifications.
- 619 7. If a `<Notification>` element is supposed to contain data about a resource (i.e., the `includeData` XML attribute
620 of a subscription has either the value `Yes` or `YesWithCommonAttributes`), the `<ItemData>` element MUST be
621 used in a `<Notification>` element. The content of an `<ItemData>` element MUST be according to the parameters
622 of the related subscription, especially `<ResultQuery>` or `<RefItem>`, and the related event which has caused this
623 `<Notification>` element to be sent inside a notification message. In the case of a change notification, the same
624 formatting rules for the content, as in the case of a query for changes, MUST be followed (see [[LibertyDST](#)],
625 Section 4 "Querying Data"). A WSP MUST NOT include any data which the WSC is not allowed to get based
626 on access rights and privacy policies defined by the resource owner. If an `<ItemData>` element should have been
627 included in a `<Notification>` element, but it is missing, the processing of the `<Notification>` element MUST fail
628 and the second level status code `MissingDataElement` SHOULD be used, in addition to the top level status
629 code, to indicate this.
- 630 8. For change notification, a `changeFormat` XML attribute MUST be added for an `<ItemData>` element to indicate
631 the format used to show the changes if a service specification has not mandated only one specific format to be
632 used for this.
- 633 9. If the data inside an `<ItemData>` element is invalid, the processing of the `<Notification>` element MUST fail and
634 the second level status code `InvalidData` SHOULD be used, in addition to the top level status code, to indicate
635 this. A WSC MUST accept all the data which can be considered as possible normal extension if extensions are
636 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.

646 6. Subscription and Notification Examples

647 6.1. Piggy-Backing a Subscription to Query

648 Consider a subscription to data that is queried.

```
649 <Query>
650   <QueryItem itemID="djkgjkdj">
651     <Select>/hp:HP/hp:AddressCard</Select>
652   </QueryItem>
653   <Subscription
654     includeData="Yes"
655     subscriptionID="tr578k-kydg4b"
656     notifyToRef="#123">
657     <RefItem itemIDRef="djkgjkdj" />
658   </Subscription>
659 </Query>
660
661
```

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

665 This subscription could later generate following notification.

```
666 <Notify>
667   <Notification subscriptionID="tr578k-kydg4b">
668     <ItemData>
669       <hp:AddressCard id="9812">
670         <hp:AddressType>urn:liberty:id-sis-hp:addrType:home </hp:AddressType>
671         <hp:Address>
672           <hp:C>us</hp:C>
673         </hp:Address>
674       </hp:AddressCard>
675     </ItemData>
676   </Notification>
677 </Notify>
678
679
```

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

682 6.2. Creating Subscription Object

683 Consider:

```
684 <Create>
685   <CreateItem objectType="_Subscription" itemID="1">
686     <NewData>
687       <Subscription
688         subscriptionID="subs123"
689         notifyToRef="#1"
690         includeData="1">
691         <ResultQuery objectType="entry">
692           <Select attributes="HELLO" />
693         </ResultQuery>
694         <RefItem itemIDRef="1" />
695       </Subscription>
696     </NewData>
697   </CreateItem>
698 </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.

706 7. Checklist for Service Specifications

- 707 1. Provide schema for **<Notify>**, **<NotifyResponse>**, and **<Subscription>** elements. If these are named differently,
708 indicate the correspondence to the standard naming.
- 709 2. If the service supports subscriptions, it will need to handle the object type "_Subscription." If the `AppDataType`
710 is defined using XML schema, this schema needs to make allowance (e.g., by using the **<xs:choice>** construct)
711 for **<Subscription>** elements by referencing the DST schema.
- 712 If the service adopts the default definition of `AppDataType`, which uses the *mixed* content model, then the default
713 is that all string data belongs to the service-specified object types while any **<Subscription>** containers belong
714 to object type "_Subscription." Further, an element of type `AppDataType` may only contain objects of one type.
- 715 3. Describe how `SelectType` applies to subscriptions.
- 716 In particular, if the service supports subscriptions, it **MUST** provide a way to specify XPath expressions for
717 querying them. The XPath expressions **MAY** be restricted to the subset described in [Section 4.6](#). This **MAY** be
718 specified by stating that "default restriction on XPaths for subscriptions applies."
- 719 4. Describe the `TriggerType` or state that is not used.
- 720 5. Describe the `AggregationType` or state that is not used.
- 721 6. Extension support.
- 722 a. If `TriggerType` or `AggregationType` is designated as unused by the service specification, then it **MAY**
723 be used for extension, provided that the extension data is
- 724 a. In URI format and use an assigned domain name as a component of the URI to ensure that extensions
725 do no collide with each other.
- 726 b. A namespace-qualified XML document.
- 727 7. Statement of how subscriptions can be established and manipulated.
- 728 a. Support CRUD manipulation of subscriptions as `objectType` "_Subscription."
729 b. Support subscribing in **<Query>**.
- 730 c. Support multiple **<Subscription>** elements in **<Query>**.
- 731 d. Support subscribing in **<Create>**.
- 732 e. Support multiple **<Subscription>** elements in **<Create>**.
- 733 f. Support subscribing in **<Modify>**.
- 734 g. Support multiple **<Subscription>** elements in **<Modify>**.
- 735 8. Start of a subscription. Usually, a subscription is valid after it has been created, but, if supported, a WSC
736 may request that a subscription is valid only after a specific time using the `starts` XML attribute. It **MUST** be
737 specified here whether the `starts` XML attribute is supported or not.
- 738 9. Subscription expiration. Usually, subscriptions expire after a certain time, but a service specification may also
739 specify, for example, that subscription expiration is not used and WSCs must cancel subscriptions after they are
740 not needed. It **MUST** be specified here whether subscriptions expire or not (e.g., Subscription expiration **MUST**
741 be used).

- 742 10. Support `expires==starts`. Is specifying the same time for both the `starts` and `expires` XML attributes to
743 request one notification message at a specified time (e.g., same value **MAY** be used both for the `starts` and the
744 `expires` XML attribute) allowed?
- 745 11. Support querying existing subscriptions. Some services or implementations may or may not support querying
746 existing subscriptions. This should be stated here (e.g., **MUST NOT** be supported).
- 747 12. Support acknowledging notifications. Some services or implementations may or may not support acknowledging
748 notifications using `<NotifyResponse>`. This should be stated here (e.g., Notifications **MUST BE** acknowledged).

749 **8. Schemata**750 **8.1. Schema for DST Reference Model with Subscriptions and**
751 **Notifications**

752 The formal schema for the reference model follows.

```
753 <?xml version="1.0" encoding="UTF-8"?>
754 <xs:schema
755     targetNamespace="urn:liberty:ssos:2006-08:ref"
756     xmlns:subsref="urn:liberty:ssos:2006-08:ref"
757     xmlns:dst="urn:liberty:dst:2006-08"
758     xmlns:subs="urn:liberty:ssos:2006-08"
759     xmlns:lu="urn:liberty:util:2006-08"
760     xmlns:xs="http://www.w3.org/2001/XMLSchema"
761     elementFormDefault="qualified"
762     attributeFormDefault="unqualified">
763   <xs:import namespace="urn:liberty:dst:2006-08"
764     schemaLocation="liberty-idwsf-dst-v2.1.xsd"/>
765   <xs:import namespace="urn:liberty:ssos:2006-08"
766     schemaLocation="liberty-idwsf-subs-v1.0.xsd"/>
767   <xs:import namespace="urn:liberty:util:2006-08"
768     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
769   <!--sec(methods)-->
770   <xs:element name="Create" type="subsref:CreateType"/>
771   <xs:element name="CreateResponse" type="subsref:CreateResponseType"/>
772   <xs:element name="Query" type="subsref:QueryType"/>
773   <xs:element name="QueryResponse" type="subsref:QueryResponseType"/>
774   <xs:element name="Modify" type="subsref:ModifyType"/>
775   <xs:element name="ModifyResponse" type="subsref:ModifyResponseType"/>
776   <xs:element name="Delete" type="subsref>DeleteType"/>
777   <xs:element name="DeleteResponse" type="subsref>DeleteResponseType"/>
778   <!--endsec(methods)-->
779   <!--sec(notifymethods)-->
780   <xs:element name="Notify" type="subsref:NotifyType"/>
781   <xs:element name="NotifyResponse" type="subsref:NotifyResponseType"/>
782   <!--endsec(notifymethods)-->
783   <!--sec(redefs)-->
784   <xs:complexType name="SelectType">
785     <xs:simpleContent>
786       <xs:extension base="xs:string"/>
787     </xs:simpleContent>
788   </xs:complexType>
789   <xs:complexType name="TestOpType">
790     <xs:simpleContent>
791       <xs:extension base="xs:string"/>
792     </xs:simpleContent>
793   </xs:complexType>
794   <xs:complexType name="SortType">
795     <xs:simpleContent>
796       <xs:extension base="xs:string"/>
797     </xs:simpleContent>
798   </xs:complexType>
799   <xs:complexType name="TriggerType">
800     <xs:simpleContent>
801       <xs:extension base="xs:string"/>
802     </xs:simpleContent>
803   </xs:complexType>
804   <xs:complexType name="AggregationType">
805     <xs:simpleContent>
806       <xs:extension base="xs:string"/>
807     </xs:simpleContent>
808   </xs:complexType>
809   <xs:complexType mixed="1" name="AppDataType">
810     <xs:sequence>
```

```

811     <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded" />
812   </xs:sequence>
813 </xs:complexType>
814 <!--endsec(redefs)-->
815 <!--sec(create)-->
816   <xs:complexType name="CreateType">
817     <xs:complexContent>
818       <xs:extension base="dst:RequestType">
819         <xs:sequence>
820           <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded" />
821           <xs:element ref="subsref:CreateItem" minOccurs="1" maxOccurs="unbounded" />
822           <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded" />
823         </xs:sequence>
824       </xs:extension>
825     </xs:complexContent>
826   </xs:complexType>
827 <!--endsec(create)-->
828 <!--sec(createaux)-->
829   <xs:element name="CreateItem" type="subsref:CreateItemType" />
830   <xs:complexType name="CreateItemType">
831     <xs:sequence>
832       <xs:element ref="subsref:NewData" minOccurs="0" maxOccurs="1" />
833     </xs:sequence>
834     <xs:attributeGroup ref="dst:CreateItemAttributeGroup" />
835   </xs:complexType>
836   <xs:element name="NewData" type="subsref:AppDataType" />
837   <xs:complexType name="CreateResponseType">
838     <xs:complexContent>
839       <xs:extension base="subsref:DataResponseType" />
840     </xs:complexContent>
841   </xs:complexType>
842   <xs:complexType name="DataResponseType">
843     <xs:complexContent>
844       <xs:extension base="dst:DataResponseBaseType" >
845         <xs:sequence>
846           <xs:element ref="subsref:ItemData" minOccurs="0" maxOccurs="unbounded" />
847         </xs:sequence>
848       </xs:extension>
849     </xs:complexContent>
850   </xs:complexType>
851 <!--endsec(createaux)-->
852 <!--sec(query)-->
853   <xs:complexType name="QueryType">
854     <xs:complexContent>
855       <xs:extension base="dst:RequestType">
856         <xs:sequence>
857           <xs:element ref="subsref:TestItem" minOccurs="0" maxOccurs="unbounded" />
858           <xs:element ref="subsref:QueryItem" minOccurs="0" maxOccurs="unbounded" />
859           <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded" />
860         </xs:sequence>
861       </xs:extension>
862     </xs:complexContent>
863   </xs:complexType>
864 <!--endsec(query)-->
865 <!--sec(queryaux)-->
866   <xs:element name="TestItem" type="subsref:TestItemType" />
867   <xs:complexType name="TestItemType">
868     <xs:complexContent>
869       <xs:extension base="dst:TestItemBaseType">
870         <xs:sequence>
871           <xs:element name="TestOp" minOccurs="0" maxOccurs="1" type="subsref:TestOpType" />
872         </xs:sequence>
873       </xs:extension>
874     </xs:complexContent>
875   </xs:complexType>
876   <xs:element name="QueryItem" type="subsref:QueryItemType" />
877   <xs:complexType name="QueryItemType">

```

```

878     <xs:complexContent>
879         <xs:extension base="subsref:ResultQueryType">
880             <xs:attributeGroup ref="dst:PaginationAttributeGroup" />
881         </xs:extension>
882     </xs:complexContent>
883 </xs:complexType>
884 <!--endsec(queryaux)-->
885 <!--sec(queryresp)-->
886 <xs:complexType name="QueryResponseType">
887     <xs:complexContent>
888         <xs:extension base="dst:DataResponseBaseType">
889             <xs:sequence>
890                 <xs:element ref="lu:TestResult" minOccurs="0" maxOccurs="unbounded" />
891                 <xs:element ref="subsref:Data" minOccurs="0" maxOccurs="unbounded" />
892             </xs:sequence>
893         </xs:extension>
894     </xs:complexContent>
895 </xs:complexType>
896 <xs:element name="Data" type="subsref:DataType" />
897 <xs:complexType name="DataType">
898     <xs:complexContent>
899         <xs:extension base="subsref:ItemDataType">
900             <xs:attributeGroup ref="dst:PaginationResponseAttributeGroup" />
901         </xs:extension>
902     </xs:complexContent>
903 </xs:complexType>
904 <!--endsec(queryresp)-->
905 <!--sec(mod)-->
906 <xs:complexType name="ModifyType">
907     <xs:complexContent>
908         <xs:extension base="dst:RequestType">
909             <xs:sequence>
910                 <xs:element ref="subsref:Subscription" minOccurs="0" maxOccurs="unbounded" />
911                 <xs:element ref="subsref:ModifyItem" minOccurs="1" maxOccurs="unbounded" />
912                 <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded" />
913             </xs:sequence>
914         </xs:extension>
915     </xs:complexContent>
916 </xs:complexType>
917 <!--endsec(mod)-->
918 <!--sec(modaux)-->
919 <xs:element name="ModifyItem" type="subsref:ModifyItemType" />
920 <xs:complexType name="ModifyItemType">
921     <xs:sequence>
922         <xs:element ref="subsref:Select" minOccurs="0" maxOccurs="1" />
923         <xs:element ref="subsref:NewData" minOccurs="0" maxOccurs="1" />
924     </xs:sequence>
925     <xs:attributeGroup ref="dst:ModifyItemAttributeGroup" />
926 </xs:complexType>
927 <xs:complexType name="ModifyResponseType">
928     <xs:complexContent>
929         <xs:extension base="subsref:DataResponseType" />
930     </xs:complexContent>
931 </xs:complexType>
932 <!--endsec(modaux)-->
933 <!--sec(del)-->
934 <xs:complexType name="DeleteType">
935     <xs:complexContent>
936         <xs:extension base="dst:RequestType">
937             <xs:sequence>
938                 <xs:element ref="subsref:DeleteItem" minOccurs="1" maxOccurs="unbounded" />
939             </xs:sequence>
940         </xs:extension>
941     </xs:complexContent>
942 </xs:complexType>
943 <xs:element name="DeleteItem" type="subsref:DeleteItemType" />
944 <xs:complexType name="DeleteItemType">
    
```

```

945     <xs:complexContent>
946     <xs:extension base="dst:DeleteItemBaseType">
947     <xs:sequence>
948     <xs:element ref="subsref:Select" minOccurs="0" maxOccurs="1"/>
949     </xs:sequence>
950     </xs:extension>
951     </xs:complexContent>
952 </xs:complexType>
953 <xs:complexType name="DeleteResponseType">
954     <xs:complexContent>
955     <xs:extension base="lu:ResponseType"/>
956     </xs:complexContent>
957 </xs:complexType>
958 <!--endsec(del)-->
959 <!--sec(resqry)-->
960 <xs:element name="Select" type="subsref:SelectType"/>
961 <xs:element name="ResultQuery" type="subsref:ResultQueryType"/>
962 <xs:complexType name="ResultQueryType">
963     <xs:complexContent>
964     <xs:extension base="dst:ResultQueryBaseType">
965     <xs:sequence>
966     <xs:element ref="subsref:Select" minOccurs="0" maxOccurs="1"/>
967     <xs:element name="Sort" minOccurs="0" maxOccurs="1" type="subsref:SortType"/>
968     </xs:sequence>
969     </xs:extension>
970     </xs:complexContent>
971 </xs:complexType>
972 <xs:element name="ItemData" type="subsref:ItemDataType"/>
973 <xs:complexType name="ItemDataType">
974     <xs:complexContent>
975     <xs:extension base="subsref:AppDataType">
976     <xs:attributeGroup ref="dst:ItemDataAttributeGroup"/>
977     </xs:extension>
978     </xs:complexContent>
979 </xs:complexType>
980 <!--endsec(resqry)-->
981 <!--sec(subscr)-->
982 <xs:element name="Subscription" type="subsref:SubscriptionType"/>
983 <xs:complexType name="SubscriptionType">
984     <xs:complexContent>
985     <xs:extension base="subs:SubscriptionType">
986     <xs:sequence>
987     <xs:element ref="subsref:ResultQuery" minOccurs="0" maxOccurs="unbounded"/>
988     <xs:element name="Aggregation" minOccurs="0" maxOccurs="1" type="subsref:AggregationType"/>
989     <xs:element name="Trigger" minOccurs="0" maxOccurs="1" type="subsref:TriggerType"/>
990     </xs:sequence>
991     </xs:extension>
992     </xs:complexContent>
993 </xs:complexType>
994 <!--endsec(subscr)-->
995 <!--sec(notif)-->
996 <xs:complexType name="NotifyType">
997     <xs:complexContent>
998     <xs:extension base="dst:RequestType">
999     <xs:sequence>
1000     <xs:element ref="subsref:Notification" minOccurs="0" maxOccurs="unbounded"/>
1001     </xs:sequence>
1002     <xs:attributeGroup ref="subs:NotifyAttributeGroup"/>
1003     </xs:extension>
1004     </xs:complexContent>
1005 </xs:complexType>
1006 <xs:element name="Notification" type="subsref:NotificationType"/>
1007 <xs:complexType name="NotificationType">
1008     <xs:complexContent>
1009     <xs:extension base="subs:NotificationType">
1010     <xs:sequence>
1011     <xs:element ref="subsref:ItemData" minOccurs="0" maxOccurs="unbounded"/>

```

```
1012     </xs:sequence>
1013   </xs:extension>
1014 </xs:complexContent>
1015 </xs:complexType>
1016 <xs:complexType name="NotifyResponseType">
1017   <xs:complexContent>
1018     <xs:extension base="subs:NotifyResponseType" />
1019   </xs:complexContent>
1020 </xs:complexType>
1021 <!--endsec(notif)-->
1022 </xs:schema>
1023
1024
```

1025 8.2. Subscriptions Utility Schema

1026 The formal utility schema follows.

```
1027 <?xml version="1.0" encoding="UTF-8"?>
1028 <xs:schema
1029   targetNamespace="urn:liberty:ssos:2006-08"
1030   xmlns:subs="urn:liberty:ssos:2006-08"
1031   xmlns:lu="urn:liberty:util:2006-08"
1032   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1033   elementFormDefault="qualified"
1034   attributeFormDefault="unqualified">
1035   <xs:import namespace="urn:liberty:util:2006-08"
1036     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1037 <!--sec(subscr)-->
1038 <xs:complexType name="SubscriptionType">
1039   <xs:sequence>
1040     <xs:element ref="subs:RefItem" minOccurs="0" maxOccurs="unbounded" />
1041     <xs:element ref="lu:Extension" minOccurs="0" maxOccurs="unbounded" />
1042   </xs:sequence>
1043   <xs:attribute name="subscriptionID" use="required" type="lu:IDType" />
1044   <xs:attribute name="notifyToRef" use="required" type="xs:anyURI" />
1045   <xs:attribute name="adminNotifyToRef" use="optional" type="xs:anyURI" />
1046   <xs:attribute name="starts" use="optional" type="xs:dateTime" />
1047   <xs:attribute name="expires" use="optional" type="xs:dateTime" />
1048   <xs:attribute name="id" use="optional" type="xs:ID" />
1049   <xs:attribute name="includeData" use="optional" />
1050   <xs:simpleType>
1051     <xs:restriction base="xs:string">
1052       <xs:enumeration value="Yes" />
1053       <xs:enumeration value="No" />
1054       <xs:enumeration value="YesWithCommonAttributes" />
1055     </xs:restriction>
1056   </xs:simpleType>
1057 </xs:attribute>
1058 </xs:complexType>
1059 <xs:element name="RefItem" type="subs:RefItemType" />
1060 <xs:complexType name="RefItemType">
1061   <xs:attribute name="subscriptionID" use="optional" type="lu:IDType" />
1062   <xs:attribute ref="lu:itemIDRef" use="required" />
1063 </xs:complexType>
1064 <!--endsec(subscr)-->
1065 <!--sec(notif)-->
1066 <xs:attributeGroup name="NotifyAttributeGroup">
1067   <xs:attribute name="timeStamp" use="optional" type="xs:dateTime" />
1068 </xs:attributeGroup>
1069 <xs:complexType name="NotificationType">
1070   <xs:sequence>
1071     <xs:element ref="lu:TestResult" minOccurs="0" maxOccurs="unbounded" />
1072   </xs:sequence>
1073   <xs:attribute name="id" use="optional" type="xs:ID" />
```

```
1074     <xs:attribute name="subscriptionID" use="required" type="lu:IDType"/>
1075     <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
1076     <xs:attribute name="endReason" use="optional" type="xs:anyURI"/>
1077 </xs:complexType>
1078 <xs:complexType name="NotifyResponseType">
1079     <xs:complexContent>
1080         <xs:extension base="lu:ResponseType"/>
1081     </xs:complexContent>
1082 </xs:complexType>
1083 <!--endsec(notif)-->
1084 </xs:schema>
1085
1086
```

1087 **References**

1088 **Normative**

- 1089 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14
1090 December, 2004). <http://www.projectliberty.org/specs>
- 1091 [LibertyDST] Kellomäki, Sampo, Kainulainen, Jukka, eds. "Liberty ID-WSF Data Services Template," Version 2.1,
1092 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1093 [LibertyDisco] Hodges, Jeff, Cahill, Conor, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0,
1094 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1095 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Lib-
1096 erty ID-WSF SOAP Binding Specification," Version 2.0, Liberty Alliance Project (30 July, 2006).
1097 <http://www.projectliberty.org/specs>
- 1098 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0,
1099 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1100 [LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version v2.0, Liberty
1101 Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 1102 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July,
1103 2006). <http://www.projectliberty.org/specs>
- 1104 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
1105 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
1106 <http://www.w3.org/TR/xmlschema-1/>
- 1107 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1108 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 1109 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
1110 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
1111 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 1112 [XPath] Clark, J., DeRose, S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0,"
1113 Recommendation, W3C <http://www.w3.org/TR/xpath> [August 2003].

1114 **Informative**

- 1115 [LibertyPeopleService] Koga, Yuzo, Madsen, Paul, eds. "Liberty ID-WSF People Service Specification," Version 1.0,
1116 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>