



Liberty ID-WSF Data Services Template Specification

Version: v2.0

Editors:

Jukka Kainulainen, Nokia Corporation
Aravindan Ranganathan, Sun Microsystems, Inc

Contributors:

Robert Aarts, Trustgenix
Rajeev Angal, Sun Microsystems, Inc.
Conor Cahill, AOL Time Warner, Inc.
Carolina Canales-Valenzuela, Ericsson
Darryl Champagne, IEEE-ISTO
Andy Feng, AOL Time Warner, Inc.
Gael Gourmelen, France Telecom
Lena Kannappan, France Telecom
Sampo Kellomaki, Symlabs, Inc.
John Kemp, Nokia Corporation
Paul Madsen, NTT
Matti Saarenpaa, Nokia Corporation
Jonathan Sergent, Sun Microsystems, Inc.
Greg Whitehead, Trustgenix

Abstract:

This specification provides protocols, schema and processing rules for the query and modification of data attributes exposed by a data service (such as a personal profile service or a geolocation service) using the Liberty Identity Web Services Framework (ID-WSF). Also subscribing to notifications related to those data attributes and sending and receiving those notifications are supported. The specification also defines some guidelines and common XML attributes and data types for data services.

Filename: liberty-idwsf-dst-v2.0.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.** Implementors
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 © 2004-2005 ADAE; Adobe Systems; America Online, Inc.; American Express Company; Avatier
16 Corporation; Axalto; Bank of America Corporation; BIPAC; Computer Associates International, Inc.; DataPower
17 Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Fidelity Investments;
18 Forum Systems, Inc. ; France Telecom; Gamefederation; Gemplus; General Motors; Giesecke & Devrient GmbH;
19 Hewlett-Packard Company; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard
20 International; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon
21 Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle
22 Corporation; Ping Identity Corporation; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp
23 Laboratories of America; Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Telefonica Moviles, S.A.;
24 Trusted Network Technologies.; Trustgenix; UTI; VeriSign, Inc.; Vodafone Group Plc. All rights reserved.

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

31 **Contents**

32 1. Overview 4
33 2. Data Model 6
34 3. Message Interface 13
35 4. Querying Data 23
36 5. Modifying Data 39
37 6. Subscriptions and Notifications 46
38 7. The Schema for the DST Protocols 59
39 8. Checklist for Service Specifications 64
40 References 71

41 1. Overview

42 This specification provides protocols for the query and modification of data attributes related to a Principal, and
43 exposed by a data service. The protocols are also provided for subscribing to notification related to those attributes
44 and sending and receiving those notifications. Additionally, some guidelines, common XML attributes and data types
45 are defined for data services.

46 This specification does not give a strict definition as which services are data services and which are not, i.e. to which
47 services this specification is targeted. A data service, as considered by this specification, is a web service that supports
48 the storage and update of specific data attributes regarding a Principal. A data service might also expose dynamic
49 data attributes regarding a Principal. Those dynamic attributes may not be stored by an external entity, but the service
50 knows or can dynamically generate their values.

51 An example of a data service would be a service that hosts and exposes a Principal's profile information (such as name,
52 address and phone number). An example of a data service exposing dynamic attributes is a geolocation service.

53 The data services using this specification can also support other protocols than those specified here. They are not
54 restricted to support just querying and modifying data attributes and subscribing notifications and sending those, but
55 they can also support actions (e.g. making reservations). Also some services might support only querying data without
56 supporting modifications and in some cases there could be services supporting only modifications without supporting
57 querying, i.e. other parties are allowed to give new data, but not query existing. The specification provides many
58 features and data services must choose, which features to use and how.

59 This specification has three main parts. First some common attributes, guidelines and type definitions to be used by
60 different data services are defined and the XML schema for those is provided. Secondly, the methods of accessing
61 the data; providing an XML schema for the Data Services Template (DST) protocols. Finally, a checklist is given for
62 writing services on top of the DST.

63 **Note:**

64 This specification does not define any XML target namespace. It provides two utility schemas to be included
65 by the data services. The Data Services Template schemas will appear in the namespace of the data services.
66 This specification uses in examples the ID-SIS Personal Profile service (see [\[LibertyIDPP\]](#)), which is built
67 on top of the DST, and the `pp` is the default namespace used in examples, but it has no other relationship
68 to the Data Services Template. Note also that the Data Services Template schemas includes Liberty Utility
69 schema and some elements and types are defined in that schema. Some type definitions are also imported
70 from [\[LibertyMetadata\]](#), [\[LibertyDisco\]](#) and [\[LibertySOAPBinding\]](#).

71 1.1. Notation

72 This specification uses schema documents conforming to W3C XML Schema (see [\[Schema1\]](#)) and normative text to
73 describe the syntax and semantics of XML-encoded protocol messages. Note: Phrases and numbers in brackets [] refer
74 to other documents; details of these references can be found at the end of this document. There are some exceptions
75 to this rule. Brackets [] are also used to indicate logical elements groups in text, where `[LogicalElement]` is used
76 instead of `<RealElement1>`, `<RealElement2>`,... Please note the different font compared to references to other
77 documents.

78 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
79 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in
80 [\[RFC2119\]](#): "they MUST only be used where it is actually required for interoperability or to limit behavior which has
81 potential for causing harm (e.g., limiting retransmissions)."

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

85 The following namespaces are used in the schema definitions:

- 86 • The prefix `xs:` stands for the W3C XML schema namespace (<http://www.w3.org/2001/XMLSchema>).
87 [\[Schema1\]](#)
- 88 • The prefix `xml:` stands for the W3C XML namespace (<http://www.w3.org/XML/1998/namespace>).
- 89 • The prefix `disco:` stands for the Liberty ID-WSF Discovery Service schema namespace
90 (<urn:liberty:disco:2003-08>). [\[LibertyDisco\]](#)
- 91 • The prefix `md:` stands for the Liberty Metadata schema namespace (<urn:liberty:metadata:2003-08>).
92 [\[LibertyMetadata\]](#)
- 93 • The prefix `sb-ext:` stands for the Liberty ID-WSF SOAP Binding Extension schema namespace
94 (<urn:liberty:sb:2004-04>). [\[LibertySOAPBinding\]](#)

95 The following namespaces are used in examples:

- 96 • The prefix `pp:` stands for the Liberty ID-SIS Personal Profile Service namespace (<urn:liberty:id-sis-pp:2003-08>).
97 [\[LibertyIDPP\]](#).
- 98 • The prefix `ds:` stands for the W3C XML signature namespace (<http://www.w3.org/2000/09/xmldsig#>).
99 [\[XMLDsig\]](#)

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

102 For readability, when an XML Schema type is specified to be `xs:boolean`, this document discusses the values as
103 "true" and "false" rather than the "1" and "0" which will exist in the document instances.

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

105 **1.2. Liberty Considerations**

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

110 2. Data Model

111 For each different type of a data service an XML schema must be specified. An example of a service type is Liberty
112 ID-SIS Personal Profile Service [[LibertyIDPPP](#)]. See [[LibertyDisco](#)] for more information about service types. The
113 XML schema of a service type specifies the data the service can host. The XML schema for a service type defines the
114 data the service type can host and the structure if this data.

115 Typically the data structure is hierarchical and has one root node. Individual branches of the structure can be accessed
116 separately and the whole structure can be accessed by pointing to the root node. A data service may also be defined so
117 that there is no one data structure to which data is stored and from which data is queried. There can be different request
118 parameters defined and a number of data sets defined to be accessed using these parameter. The service specification
119 defines, how the defined parameters are used and which are the data sets.

120 The data may be stored in implementation-specific ways, but will be exposed by the service using the XML schema
121 specified both in this document, and that of the defined service type. This also means that the XML document defined
122 by the schema is a conceptual XML document. Depending upon the implementation, there may be no XML document
123 that matches the complete conceptual document. The internal storage of the data is separate and distinct from the
124 document published through this model.

125 The schemas for different service types may have common characteristics. This section describes the commonalities
126 specified by the Data Services Template, provides schema for common attributes and data types, and also gives some
127 normative guidelines.

128 2.1. Guidelines for Schemas

129 The schemas of different data services SHOULD follow guidelines defined here. The purpose of these guidelines is to
130 make the use of the Data Services Template easier when defining and implementing services.

- 131 • Each data attribute regarding the Principal SHOULD be defined as an XML element of a suitable type.
- 132 • XML attributes SHOULD be used only to qualify the data attribute defined as XML elements and not contain the
133 actual data values related to the Principal.
- 134 • An XML element SHOULD either contain other XML elements or actual data value. An XML element SHOULD
135 NOT have mixed content, i.e. both a value and sub-elements. Also complex types all and choice SHOULD
136 NOT be used.
- 137 • Once a data attribute has been published in a specification for a service type, its syntax and semantics MUST not
138 change. If evolution in syntax or semantics is needed, any new version of a data attribute MUST be assigned a
139 different name, effectively creating a new attribute with new semantics so that it does not conflict with the original
140 attribute definition.
- 141 • All elements MUST be defined as global elements, when they can be requested individually. When elements with
142 complex type are defined, references to global elements are used. The reason for this guideline is that the XML
143 Schema for a service does not only define the syntax of the data supported by the service but also the transfer
144 syntax. In many cases it should be possible to query and modify individual elements.
- 145 • The type definitions provided by the XML Schema SHOULD be used, when they cover the requirements.

146 **2.2. Extending a Service**

147 A service defined by its specification and schema MAY be extended in different ways. What type of extensions are
148 supported in practice MUST be specified individually for each service type in a specification for that service type.

- 149 • An implementation MAY add new elements and attributes to the specified schema. These new elements and
150 attributes MUST use their own XML namespace until they are adopted by the official Liberty specification and
151 schema of the service type.
- 152 • When new features for a service are specified (e.g. new elements), new keywords SHOULD be specified for
153 indicating the new features using the `<Option>` element (see [\[LibertyDisco\]](#) for more information).
- 154 • New values for enumerators MAY be specified subsequent to the release of a specification document for a
155 specific service type. The specification for a service type MUST specify the authority for registering new official
156 enumerators (whether that authority is the specification itself, or some external authority). For specification done
157 by Liberty Alliance, see [\[LibertyReg\]](#).
- 158 • Elements defined in the XML schema for a service type MAY contain an `<xs:any>` element to support ar-
159bitrary schema extension. When the `<xs:any>` elements are in the schema, an implementation MAY sup-
160port this type of extension, but is not required to. The `<xs:any>` elements SHOULD always be put inside
161 `<Extension>` elements. If an implementation does support this type of schema extension, then it MAY regis-
162ter `urn:liberty:dst:can:extend` discovery option keyword. When a service holds new data, which is not
163 defined in the schema for the service type but is stored using this kind of support for extensions, it MAY register
164 `urn:liberty:dst:extend` discovery option keyword.

165 **2.3. Time Values and Synchronization**

166 Some of the common XML attributes are time values. All Liberty time values have the type `dateTime`, which is built
167 in to the W3C XML Schema Datatypes specification. Liberty time values MUST be expressed in UTC form, indicated
168 by a "Z" immediately following the time portion of the value.

169 Liberty requesters and responders SHOULD NOT rely on other applications supporting time resolution finer than sec-
170 onds, as implementations MAY ignore fractional second components specified in timestamp values. Implementations
171 MUST NOT generate time instants that specify leap seconds.

172 The timestamps used in the DST schemas are only for the purpose of data synchronization and no assumptions should
173 be made as to clock synchronization.

174 **2.4. Common Attributes**

175 The XML elements defined in the XML schemas for the services either contain data values or other XML elements.
176 So an XML element is either a leaf element or a container. The containers do not have any other data content than
177 other XML elements and possible qualifying XML attributes. The other type of XML elements are considered *leaf*
178 elements, and as such, do not contain other XML elements. These leaf elements can be further divided into two
179 different categories: normal and localized. The localized leaf elements contain text using a local writing system.

180 Both leaf and container XML elements can have service-specific XML attributes, but there are also common XML
181 attributes supplied for use by all data services. These common XML attributes are technical attributes, which are
182 usually created by the Web Service Provider (WSP) hosting a data service (for more details, see [Section 5](#)). These
183 technical attributes are not mandatory for all data services, but if they are implemented, they MUST be implemented in
184 the way described in this document. Each service should specify separately if one or more of these common attributes
185 are mandatory or optional for that service. In addition to the common attributes, we define attribute groups containing

186 these common attribute groups. There are three attribute groups, one common (`commonAttributes`) mainly targeted
187 for container elements and two for the leaf elements (`leafAttributes` and `localizedLeafAttributes`).

188 **2.4.1. The commonAttributes Attribute Group**

189 There are only two common attributes:

190 `id` [Optional]

191 The `id` is a unique identifier within a document. It can be used to refer uniquely to an element, especially
192 when there may be several XML elements with the same name. If the schema for a data service does not
193 provide any other means to distinguish between two XML elements and this functionality is needed, the `id`
194 attribute **MUST** be used. This `id` attribute is only meant for distinguishing XML elements within the same
195 conceptual XML document. It **MUST NOT** be used for globally unique identifiers, because that would create
196 privacy problems. An implementation **MAY** set specific length restrictions on `id` attributes to enforce this.
197 The value of the `id` attribute **SHOULD** stay the same when the content of the element is modified so the same
198 value of the `id` attribute can be used when querying the same elements at different times. The `id` attribute
199 **MUST NOT** be used for storing any data and it **SHOULD** be kept short.

200 `modificationTime` [Optional]

201 The `modificationTime` specifies the last time that the element was modified. Modification includes chang-
202 ing either the value of the element itself, or any sub-element. So the time of the modification **MUST** be prop-
203 agated up all the way to the root element, when container elements have the `modificationTime` attribute.
204 If the root element has the `modificationTime` attribute, it states the time of the latest modification. Note
205 that a data service may have the `modificationTime` attribute used only in leaf elements or not even for
206 those as it is optional.

207 **2.4.2. The leafAttributes Attribute Group**

208 This group includes the `commonAttributes` attribute group and defines three more attributes for leaf elements:

209 `modifier` [Optional]

210 The `modifier` is the `ProviderID` (see [\[LibertyMetadata\]](#)) of the service provider which last modified the
211 data element.

212 `ACC` [Optional]

213 The acronym `ACC` stands for *attribute collection context* which describes the context (or mechanism) used
214 in collecting the data. This might give useful information to a requester, such as whether any validation has
215 been done. The `ACC` always refers to the current data values, so whenever the value of an element is changed,
216 the value of the `ACC` must be updated to reflect the new situation. The `ACC` is of type **anyURI**.

217 The following are defined values for the `ACC` attribute:

- 218 • `urn:liberty:dst:acc:unknown`

219 This means that there has been no validation, or the values are just voluntary input from the user. The `ACC` **MAY**
220 be omitted in the message exchange when it has this value, as this value is equivalent to supplying no `ACC` attribute
221 at all.

- 222 • `urn:liberty:dst:acc:incentive`

223 There has been some incentive for user to supply correct input (such as a gift sent to the user in return for their
224 input).

- 225 • `urn:liberty:dst:acc:challenge`
226 A challenge mechanism has been used to validate the collected data (e.g. an email sent to address and a reply
227 received or an SMS message sent to a mobile phone number containing a WAP URL to be clicked to complete the
228 data collection)
- 229 • `urn:liberty:dst:acc:secondarydocuments`
230 The value has been validated from secondary documents (such as the address from an electric bill).
- 231 • `urn:liberty:dst:acc:primarydocuments`
232 The value has been validated from primary documents (for example, the name and identification number from a
233 passport).
- 234 Other values are allowed for `ACC`, but this specification normatively defines usage only for the values listed
235 above.
236 When the `ACC` is included in the response message, the response SHOULD be signed by the service provider
237 hosting the data service.
- 238 `ACCTime` [Optional]
239 This defines the time that the value for the `ACC` attribute was given. Note that this can be different from the
240 `modificationTime`. The `ACC` contains information that may be related to the validation of the entry. Such
241 validation might happen later than the time the entry was made, or modified. The entry can be validated more
242 than once.

243 2.4.3. The `localizedLeafAttributes` Attribute Group

244 This attribute group includes the `leafAttributes` attribute group and defines two more attributes to support localized
245 data, when the Latin 1 character set is not used:

- 246 `xml:lang` [Required]
247 This defines the language used for the value of a localized leaf element. When the
248 `localizedLeafAttributes` attribute group is used for an element, this is a mandatory XML attribute.
- 249 `script` [Optional]
250 Sometimes the language does not define the writing system used. In such cases, this attribute defines
251 the writing system in more detail. This specification defines the following values for this attribute:
252 `urn:liberty:dst:script:kana` and `urn:liberty:dst:script:kanji`. See [LibertyReg] where to
253 find more values and how to specify more values.

254 2.4.4. Individual common attributes

255 In addition to the previous attribute groups a couple of more common attributes are defined and available for services.
256 The attributes in attribute groups can also be used individually without taking the whole attribute group into use, but
257 the following attributes are assumed to be more seldomly used and so they are not included in any of the attribute
258 groups to keep those attribute groups more common.

- 259 `refreshOnOrAfter`
260 A WSC may cache the information in the element and if it chooses to do so, it SHOULD refresh the data from
261 the WSP if it attempts to use the data beyond the time specified. If the data is not refreshed (for whatever
262 reason) a WSC MAY continue to use it. This parameter does NOT place an obligation upon the WSP to keep
263 the value of the data static during this timespan, so it is possible (and in some cases likely) that the contents
264 of the element will change during the specified period. WSCs that require timely data should request the

265 most up to date data when they need it rather than cacheing the data (or a WSC may subscribe for data change
266 notifications).

267 `destroyOnOrAfter`

268 Even if a WSC has not been able to refresh the information, it SHOULD destroy it, if the element containing
269 the information has the attribute `destroyOnOrAfter` and the time specified by that attribute has come. The
270 information most probably is so out of date that it is unusable.

271 2.5. Common Data Types

272 The type definitions provided by XML schema can not always be used directly by Liberty ID-WSF data services, as
273 they lack the common attributes noted above. The DST data type schema (Section 2.6) provides types derived from
274 the XML Schema ([XML]) datatype definitions with those common attributes added to the type definitions. Please
275 note that for strings there are two type definitions, one for localized elements and another for elements normalized
276 using the Latin 1 character set.

277 The following type definitions are provided:

278 • `DSTLocalizedString`

279 • `DSTString`

280 • `DSTInteger`

281 • `DSTURI`

282 • `DSTDate`

283 • `DSTMonthDay`

284 2.6. The Schema for Common XML Attributes and Data Types

```
285
286     <?xml version="1.0" encoding="UTF-8"?>
287 <xs:schema xmlns:md="urn:liberty:metadata:2003-08"
288   xmlns:xs="http://www.w3.org/2001/XMLSchema"
289   elementFormDefault="qualified" attributeFormDefault="unqualified">
290
291   <xs:import namespace="urn:liberty:metadata:2003-08"
292     schemaLocation="liberty-metadata-v1.1.xsd"/>
293   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
294     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
295   <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>
296   <xs:annotation>
297     <xs:documentation>Liberty Alliance Project ID-WSF Data Services Template Data Types_
298 Schema</xs:documentation>
299     <xs:documentation>
300 The source code in this XSD file was excerpted verbatim from:
301
302 Liberty ID-WSF Data Services Template Specification
303 Version 1.1
304 14 December 2004
305
306 Copyright (c) 2004 Liberty Alliance participants, see
307 http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php
308
309     </xs:documentation>
310   </xs:annotation>
```

```
311 <!-- Common attributes to be used by different services when found useful/needed-->
312 <xs:attribute name="id" type="IDType" />
313 <xs:attribute name="modificationTime" type="xs:dateTime" />
314 <xs:attributeGroup name="commonAttributes">
315   <xs:attribute ref="id"/>
316   <xs:attribute ref="modificationTime" />
317 </xs:attributeGroup>
318 <xs:attribute name="ACC" type="xs:anyURI" />
319 <xs:attribute name="ACCTime" type="xs:dateTime" />
320 <xs:attribute name="modifier" type="md:entityIDType" />
321 <xs:attributeGroup name="leafAttributes">
322   <xs:attributeGroup ref="commonAttributes" />
323   <xs:attribute ref="ACC" />
324   <xs:attribute ref="ACCTime" />
325   <xs:attribute ref="modifier" />
326 </xs:attributeGroup>
327 <xs:attribute name="script" type="xs:anyURI" />
328 <xs:attributeGroup name="localizedLeafAttributes">
329   <xs:attributeGroup ref="leafAttributes" />
330   <xs:attribute ref="xml:lang" use="required" />
331   <xs:attribute ref="script" />
332 </xs:attributeGroup>
333 <!-- Common data types to be used by different services when found useful/needed-->
334 <xs:complexType name="DSTLocalizedString">
335   <xs:simpleContent>
336     <xs:extension base="xs:string">
337       <xs:attributeGroup ref="localizedLeafAttributes" />
338     </xs:extension>
339   </xs:simpleContent>
340 </xs:complexType>
341 <xs:complexType name="DSTString">
342   <xs:simpleContent>
343     <xs:extension base="xs:string">
344       <xs:attributeGroup ref="leafAttributes" />
345     </xs:extension>
346   </xs:simpleContent>
347 </xs:complexType>
348 <xs:complexType name="DSTInteger">
349   <xs:simpleContent>
350     <xs:extension base="xs:integer">
351       <xs:attributeGroup ref="leafAttributes" />
352     </xs:extension>
353   </xs:simpleContent>
354 </xs:complexType>
355 <xs:complexType name="DSTURI">
356   <xs:simpleContent>
357     <xs:extension base="xs:anyURI">
358       <xs:attributeGroup ref="leafAttributes" />
359     </xs:extension>
360   </xs:simpleContent>
361 </xs:complexType>
362 <xs:complexType name="DSTDate">
363   <xs:simpleContent>
364     <xs:extension base="xs:date">
365       <xs:attributeGroup ref="leafAttributes" />
366     </xs:extension>
367   </xs:simpleContent>
368 </xs:complexType>
369 <xs:complexType name="DSTMonthDay">
370   <xs:simpleContent>
371     <xs:extension base="xs:gMonthDay">
372       <xs:attributeGroup ref="leafAttributes" />
373     </xs:extension>
374   </xs:simpleContent>
375 </xs:complexType>
376 </xs:schema>
377
```

378

379 3. Message Interface

380 This specification defines number of protocols for data services. These protocols rely mainly on a request/response
 381 message-exchange pattern. The only exception is the notification messages, which might not get any response. The
 382 messages specified in this document are carried in the SOAP body. No additional content is specified for the SOAP
 383 header in this document, but implementers of these protocols MUST follow the rules defined in [LibertySOAPBind-
 384 ing].

385 The following two tables list the protocol elements specified by this specification.

386 **Table 1. Request/Response**

| Request by a WSC | Response by a WSP |
|----------------------|---------------------|
| <Query> | <QueryResponse> |
| <Modify> | <ModifyResponse> |
| <Subscribe> | <SubscribeResponse> |
| <QuerySubscriptions> | <Subscriptions> |

387 **Table 2. Notification/Acknowledgement**

| Notification by a WSP | Acknowledgement by a WSC |
|-----------------------|--------------------------|
| <Notify> | <NotifyResponse> |
| <Ended> | <EndedResponse> |

388 The messages for different protocols have common features, attributes and elements. These common issues are dis-
 389 cussed in this chapter and the actual messages are specified in the following chapters. Together with common parts the
 390 related processing rules are also defined. In the text especially in the processing rules the [RequestElement] is used
 391 to replace the actual request element in many cases. These parts MUST be read as instead of a [RequestElement]
 392 there would be any of the following elements: <Query>, <Modify>, <Subscribe> or <QuerySubscriptions>.

393 The [ResponseElement] is used instead of the actual response element in many places. Those parts MUST be
 394 read as instead of a [ResponseElement] there would be any of the following elements: <QueryResponse>,
 395 <ModifyResponse>, <SubscribeResponse> or <Subscriptions>. Also the [NotificationElement] and
 396 the <AcknowledgementElement> are used. When these are used, the text MUST be read as instead of a
 397 [NotificationElement] there would be either of the elements <Notify> or <Ended> and instead of an
 398 [AcknowledgementElement] there would be either of the elements <NotifyResponse> or <EndedResponse>.

399 3.1. Status and Fault Reporting

400 Two mechanism are defined to report back to the requestor whether the processing of a request was successful or not
 401 or something between. [LibertySOAPBinding] defines the ID-* Fault message, which is used to convey processing

402 exception. An ordinary ID-* Message carrying normal response is used to report back application statuses including
403 normal error conditions, when an application has detected an error condition as part of the normal processing e.g.
404 processing according to the processing rules specified in this document.

405 From the Data Service Template point of view there are the following four cases in which the ID-* Fault Message is
406 used.

407 When a WSP does not recognize any [RequestElement] in the SOAP Body, it MUST return an ID-* Fault Message
408 and use IDStarMsgNotUnderstood as the value of the code attribute as specified by [LibertySOAPBinding]. In
409 the same way a WSC that receives an empty or malformed notification MUST return an ID-* Fault Message and use
410 IDStarMsgNotUnderstood as the value of the code attribute.

411 When a WSP receives a request message, which it does not support, it MUST return an ID-* Fault Message and use
412 ActionNotSupported as the value of the code attribute.

413 If a WSP based on identifying the requesting party notices that the requesting party is not allowed to make any requests,
414 it MUST return an ID-* Fault Message and use ActionNotAuthorized as the value of the code attribute.

415 A receiving party may also encounter an unexpected error due to which it fails to handle the message body. In that
416 kind of a case it MUST return an ID-* Fault Message and use UnexpectedError as the value of the code attribute.
417 A service specification MAY define more cases in which ID-* Fault Message is used.

418 Even if the processing of some parts of a message body fails, a WSP SHOULD always try to process the message body
419 as well as it cans according the specified processing rules and return normal response message indicating the failed
420 parts in returned status codes (see Section 3.1.1) as one message may contain multiple task requests and succeeding
421 in individual tasks is valuable, if the processing rules do not specify that after the first failed part the whole message
422 should fail. One request message may contain one or more [RequestElement]. One [RequestElement] may also
423 contain number of individual task request (e.g. inside a <Query> there can be multiple <QueryItem> elements). So
424 after failing to complete one requested tasks there could be a number of other tasks requested in the same message and
425 a WSP SHOULD try to complete those unless service specific processing rules specify otherwise.

426 3.1.1. <Status> element

427 A [ResponseElement] element and an <AcknowledgementElement> element contains one <Status> elements
428 to indicate whether or not the processing of a [RequestElement] element or [NotificationElement] element
429 has succeeded. The <Status> element is included from the Liberty Utility Schema. A <Status> element MAY
430 contain other <Status> elements providing more detailed information. A <Status> element has a code attribute,
431 which contains the return status as a string. The local definition of these codes is specified in this document.

432 This specification defines the following status codes to be used as values for the code attribute:

- 433 • ActionNotAuthorized
- 434 • ActionNotSupported
- 435 • AllReturned
- 436 • ChangeHistoryNotSupported
- 437 • ChangedSinceReturnsAll
- 438 • DataTooLong
- 439 • ExistsAlready

- 440 • ExtensionNotSupported
- 441 • Failed
- 442 • FormatNotSupported
- 443 • InvalidData
- 444 • InvalidExpires
- 445 • InvalidEndedTo
- 446 • InvalidEndpoint
- 447 • InvalidResourceID
- 448 • InvalidSelect
- 449 • InvalidSetID
- 450 • InvalidSetReq
- 451 • InvalidSort
- 452 • InvalidSubscriptionID
- 453 • MissingCredentials
- 454 • MissingDataElement
- 455 • MissingEndpointElement
- 456 • MissingExpiration
- 457 • MissingItemID
- 458 • MissingNewDataElement
- 459 • MissingNotifyToElement
- 460 • MissingResourceIDElement
- 461 • MissingSecurityMechIDElement
- 462 • MissingSelect
- 463 • MissingSubscriptionID
- 464 • ModifiedSince
- 465 • NewOrExisting
- 466 • NoMoreElements
- 467 • NoMultipleAllowed
- 468 • NoMultipleResources
- 469 • OK

470 • `PaginationNotSupported`

471 • `Partial`

472 • `RequestedPaginationNotSupported`

473 • `RequestedSortingNotSupported`

474 • `SecurityMechIDNotAccepted`

475 • `SetOrNewQuery`

476 • `SortNotSupported`

477 • `StaticNotSupported`

478 • `TimeOut`

479 • `TriggerNotSupported`

480 • `TypeNotSupported`

481 • `UnexpectedError`

482 • `UnspecifiedError`

483 The `<Status>` element may contain other `<Status>` elements supplying more detailed return status information.

484 The `code` attribute of the top level `<Status>` element **MUST** contain one of the following values `OK`, `Partial`

485 or `Failed`. The remainder of the values above are used to indicate more detailed return status inside second level

486 `<Status>` element(s).

487 OK
488 The value OK means that the processing of a [RequestElement] element or a [NotificationElement]
489 element has succeeded. A second level status code MAY be used to indicate some special cases, but the
490 processing of a [RequestElement] element or a [NotificationElement] element has succeeded.

491 Partial
492 The value Partial means that the processing has succeeded only partially and partially failed, e.g. in
493 the processing of a <Query> element some <QueryItem> element has been processed successfully, but
494 the processing of some other <QueryItem> elements has failed. When the value Partial is used for
495 the code attribute of the top level <Status> element, the top level <Status> element MUST have
496 second level <Status> element(s) to indicate the failed part(s) of a [RequestElement] element or a
497 [NotificationElement] element. The processing of the part(s) not referred to by any of the second
498 level <Status> elements MUST have succeeded. A WSP MUST NOT use the value Partial, if it has not
499 processed the whole [RequestElement] element or [NotificationElement] element.
500 A WSP MUST NOT use the value Partial in case of modification requests, when a failed
501 <Modification> element didn't have a valid itemID attribute, i.e. a WSP is not able to indicate
502 the failed <Modification> element. In those cases a WSP MUST use the value Failed and anything
503 changed based on the already processed part MUST be rolled back.
504 A WSP MAY also choose to fail completely another type of [RequestElement], when only a part of it
505 has failed, if the failed part does not have a valid itemID attribute. When ever the top level value Failed
506 is used instead of Partial due to one or more missing itemID attribute(s), the second level status code
507 MissingItemID MUST be used in addition to any other second level status code.
508 In some cases the most descriptive second level status code may not be used as it e.g. might compromise the
509 privacy of a Principal. In those cases, when the second level status code must be used to indicate the failed
510 parts in a case of a partial failure, the value UnspecifiedError MUST be used for the second level status
511 code.

512 Failed
513 The value Failed means that the processing of a [RequestElement] element or a
514 [NotificationElement] element has failed. Either the processing of the whole [RequestElement]
515 element or [NotificationElement] element has totally failed or it might have succeeded partially, but
516 the WSP decided to fail it completely. A specification for a service MAY also deny the use of the partial
517 failure and so force a WSP to fail, even when it could partially succeed. A second level status code SHOULD
518 be used to indicate the reason for the failure.

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

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

528 When the reference is made using the value of an id attribute, the WSP MUST check that the request did not contain
529 any itemID attribute with the same value. If there is an itemID attribute with the same value as the id attribute of
530 the offending element (or the closest ancestor in case the offending element did not have any id or itemID attributes),
531 the reference MUST NOT be made using the value of this id attribute to make sure that the reference is clear.

532 3.2. Linking with ids

533 Different types of id attributes are used to link queries and responses and notifications and acknowledgements together.
534 Response and acknowledgement messages are correlated with requests and notifications using messageID and

535 `refToMessageID` attributes that are present in the `<Correlation>` Header Block (see [\[LibertySOAPBinding\]](#)). A
536 WSC MUST include the `messageID` attribute in each request it sends and a WSP MUST include both the `messageID`
537 and the `refToMessageID` attributes in each response it sends. Similarly a WSP MUST include the `messageID`
538 attribute in each notification it sends and a WSC MUST include both the `messageID` and the `refToMessageID`
539 attributes in each notification acknowledgement it sends. Use of these attributes MUST follow the processing
540 rules specified in [\[LibertySOAPBinding\]](#). Inside messages `itemID` and `itemIDRef` attributes are used for linking
541 information inside response and acknowledgement messages to the details of request and notification messages.

542 See the definitions and the processing rules of the protocol elements and the processing rules in [Section 3.3](#) for more
543 detailed information.

544 Some elements in all messages can have `id` attributes of type `xs:ID`. These `id` attributes are necessary when some
545 part of the message points to those element. As an example, if usage directives are used, then the usage directive
546 element must point to the correct element (see [\[LibertySOAPBinding\]](#)). Some parts of the messages may be signed
547 and the `id` attribute is necessary to indicate which elements are covered by a signature.

548 3.3. Resources

549 All DST protocols have a defined hierarchy for addressing the data to be accessed. In the first level the desired resource
550 is selected. For example, a resource might be the personal profile of a certain person.

551 Multiple resources can be accessed in a single request, but different type of request MUST NOT be mixed in one
552 request message, e.g. querying and modifying can not be used simultaneously in the same request message. For each
553 resource there is one `[RequestElement]` element in a request message. Inside this element there is another element
554 identifying the resource. This identifying element is either a `<ResourceID>` element or a `<EncryptedResourceID>`
555 element. The type definitions for both elements are imported from the Liberty ID-WSF Discovery Service schema.
556 For more information about resources, different types of resource identifiers and encryption of resource identifiers see
557 [\[LibertyDisco\]](#).

558 The `ResourceIDGroup` schema is shown below:

```
559 <xs:element name="ResourceID" type="disco:ResourceIDType" />  
560 <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType" />  
561 <xs:group name="ResourceIDGroup">  
562   <xs:choice>  
563     <xs:element ref="ResourceID" />  
564     <xs:element ref="EncryptedResourceID" />  
565   </xs:choice>  
566 </xs:group>  
567  
568
```

569 When the `<ResourceID>` element would have the value `urn:liberty:isf:implied-resource` (see [\[Liberty-](#)
570 [Disco\]](#)), the element MAY be left out of the containing `[RequestElement]` element. In all other cases either the
571 `<ResourceID>` element or the `<EncryptedResourceID>` element MUST be present. See [\[LibertyPAOS\]](#) for exam-
572 ples of when the value `urn:liberty:isf:implied-resource` can be used.

573 **Note:**

574 When `<EncryptedResourceID>` element is used, the encrypted `<ResourceID>` element inside it is in the
575 namespace of [\[LibertyDisco\]](#), when the `<EncryptedResourceID>` element itself is in the namespace of the
576 data service in question as well as all other elements defined in this specification.

577 3.3.1. Common processing rules

578 A request message can contain multiple `[RequestElement]` elements as well as a notification message can contain
579 multiple `[NotificationElement]` elements. The following rules specify how those must be supported and handled:

- 580 • A receiver of a message (WSP for request messages and WSC for notification messages) MUST support one
581 [RequestElement] element inside a request message and one [NotificationElement] element inside a
582 notification message and SHOULD support multiple. If a receiver supports only one [RequestElement] or
583 [NotificationElement] element inside a request/notification message and the message contains multiple
584 [RequestElement] or [NotificationElement] elements, the processing of the whole message MUST fail
585 and a status code indicating the failure MUST be returned in the response. A more detailed status code with the
586 value NoMultipleResources SHOULD be returned in addition to the top level status code as it is not possible
587 to handle multiple resources in one message. If a WSP supports accessing multiple resources, it MAY register
588 urn:liberty:dst:multipleResources discovery option keyword.
- 589 • If a request message contains multiple [RequestElement] elements or a notification mes-
590 sage contains multiple [NotificationElement] elements, the sender MUST add itemID at-
591 tributes for each [RequestElement]/[NotificationElement] element. The receiver MUST link
592 the [ResponseElement]/<AcknowledgementElement> elements it is sending to corresponding
593 [RequestElement]/[NotificationElement] elements it received using the itemIDRef attributes, if
594 there were itemID attributes in the [RequestElement]/[NotificationElement] elements and there were
595 multiple [RequestElement]/[NotificationElement] elements in the request message.
596 If multiple [RequestElement] or [NotificationElement] elements are used in a message, a receiver
597 MUST discard any [RequestElement] or [NotificationElement] element not having a valid itemID
598 attribute. For that [RequestElement] or [NotificationElement] element no <ResponseElement> or
599 [AcknowledgementElement] element is returned as it is not possible to indicate for which [RequestElement]
600 or [NotificationElement] element that element is returned.
601 The itemIDRef attribute in a [ResponseElement]/<AcknowledgementElement> element MUST have the
602 same value as the itemID attribute in the corresponding [RequestElement]/[NotificationElement] ele-
603 ment.
- 604 • If the processing of a [RequestElement]/[NotificationElement] fails for some reason, any other
605 [RequestElement]/[NotificationElement] elements included in the same message SHOULD be processed
606 normally, as if the error had not occurred as different [RequestElement]/[NotificationElement] elements
607 inside the same message are usually independent. When processing of a [RequestElement]/[NotificationElement]
608 fails completely, the top level status code Failed MUST be used to indicate the failure and a more detailed status
609 code SHOULD be used to indicate more detailed status information. If it is possible and allowed by the service
610 specification to process successfully independent part(s) of the [RequestElement]/[NotificationElement]
611 the top level status code Partial MUST be used as specified earlier. A successful request/notification MUST be
612 indicated using the top level status code OK. Note: even with top level status code OK second level status codes
613 MAY be used.
- 614 A WSP must know which resource a WSC wants to access to be able to process the query. The following rules apply
615 to resource identifiers:
- 616 • If there is no <ResourceID> or <EncryptedResourceID> element in the [RequestElement], the processing
617 of the whole [RequestElement] MUST fail and a status code indicating failure MUST be returned in the re-
618 sponse, unless the <ResourceID> element would have had the value urn:liberty:isf:implied-resource
619 (see [LibertyDisco]). In this case the <ResourceID> MAY be left out. When either the <ResourceID> or
620 the <EncryptedResourceID> element should have been present, a more detailed status code with the value
621 MissingResourceIDElement SHOULD be used in addition to the top level status code.
- 622 • If the resource identified in the <ResourceID> or <EncryptedResourceID> element does not exist, the
623 processing of the whole [RequestElement] MUST fail and a status code indicating the failure MUST be
624 returned in the response. A more detailed status code with the value InvalidResourceID SHOULD be used
625 in addition to the top level status code.

626 **3.4. <Select> element**

627 The second level of the selection is deeper inside the [RequestElement] element. The request message must
628 describe in more detail what it wants to access inside the specified resource. This is specified in <Select> elements.

629 As an example, when the resource is a personal profile, the <Select> can point to a home address. In the case of a
630 <Query>, this means that the whole home address is requested, or for a <Modify>, the whole home address is being
631 modified, etc. When only a part of a home address is accessed, the <Select> element must point only to that part,
632 or in the case of a <Modify> the parts not to be modified must be rewritten using their existing values, when whole
633 home address is given. Different parts of the resource can be accessed using the same [RequestElement] element
634 as those elements can contain multiple <Select> elements in their own sub-structure.

635 Please note that the previous paragraph only described an example. The <Select> element may also be used
636 differently. It is defined to contain needed parameters, but the parameters are defined by the specification for a service
637 type. A service may have multiple different type of parameters characterizing data the be accessed and e.g. instead
638 of pointing to some point in a data structure, the content of the <Select> element may e.g. list the data items to be
639 accessed with some quality requirements for the data to be returned.

640 The he <Select> element may also be omitted from a request, when all the data of a resource is accessed, e.g. queried
641 or modified, in one request.

642 The type of <Select> is SelectType. Although the type is referenced by *this* specification, the type may vary
643 according to the service specifications using this schema, and therefore **MUST** be defined within each service schema.
644 As the type of the <Select> element may be quite different in different services, a service specification **MUST** specify
645 the needed processing rules, if the processing rules provided by this specification are not adequate. If there are any
646 conflicts the processing rules in the service specifications **MUST** override the processing rules in this specification.

647 When the SelectType is specified for a service, it must be very careful about what type of queries and modifies
648 needs to be supported. Typically the <Select> points to some place(s) in the conceptual XML document and it is
649 **RECOMMENDED** that a string containing an XPATH expression is used for <Select> element in those kind of
650 cases. There are many other type of cases and the SelectType must be properly specified to cover the needs of a
651 service type.

652 When XPATH is used, it is not always necessary to support full XPATH. Services **SHOULD** limit the required
653 set of XPATH expressions in their specifications when full XPATH is not required. E.g. the type and the
654 values required to be supported for the <Select> element by the ID-Personal Profile service are specified in
655 [\[LibertyIDPPP\]](#). A service may support full XPATH even if it is not required. In that case the service **MAY** register
656 the urn:liberty:dst:fullXPath discovery option keyword. If the required set of XPath expressions does not
657 include the path to each element, a service may still support all paths without supporting full XPath. In that case the
658 service **MAY** register the urn:liberty:dst:allPaths discovery option keyword.

659 **3.4.1. Common processing rules**

660 The following rules specify how the <Select> element should be processed and interpreted:

- 661 • If the <Select> element is missing from a subelement of a [RequestElement] element, when it is supposed to
662 be use, the processing of that subelement **MUST** fail and a status code indicating the failure **MUST** be returned
663 in the response. A more detailed status code with the value MissingSelect **SHOULD** be used in addition to
664 the top level status code. The subelements referred here are the <QueryItem>, the <Modification> and the
665 <Subscriptions>. All these elements are defined later with other protocol elements. Note: in some cases the
666 <Select> element is not needed

- 667 • If the `<Select>` element has invalid content, e.g. contains an invalid pointer to a data not supported by the WSP
668 or doesn't contain the specified parameters, the processing of the subelement containing the `<Select>` element
669 MUST fail and a status code indicating failure MUST be returned in the response. A more detailed status code with
670 the value `InvalidSelect` SHOULD be used in addition to the top level status code, unless a service specification
671 specifies more detailed status codes better suited for the case. Note that a data service may support extensions,
672 making it difficult for a requester to know the exact set of allowable values for the `<Select>` element.

673 3.5. The `timeStamp` Attribute

674 A response and a notification message can also have a time stamp. This time stamp is provided so that the receiving
675 party can later check whether there have been any changes since a response or a notification was received, or make
676 modifications, which will only succeed if there have been no other modifications made after the time stamp was
677 received.

678 3.5.1. Common processing rules

- 679 • A WSP MUST add a `timeStamp` to a `[ResponseElement]`, if the processing of the `[RequestElement]`
680 was successful and a WSP supports either the `changedSince` attribute or the `notChangedSince` attribute or
681 both properly. A WSP MUST also add a `timeStamp` to a `[NotificationElement]`, it supports either the
682 `changedSince` attribute or the `notChangedSince` attribute or both properly. The `timeStamp` attribute MUST
683 have a value which can also be used as a value for the `changedSince` attribute, when querying changes made after
684 the request for which the `timeStamp` was returned or the notification, which had the `timeStamp`. The value of the
685 `timeStamp` attribute MUST also be such that it can be used as a value for the `notChangedSince` attribute, when
686 making modifications after the request for which the `timeStamp` was returned or after receiving the notification
687 message, which carried the `timeStamp` and the modifications will not succeed, if there has been any modification
688 after this request/notification.

689 3.6. The `<Extension>` Element

690 All messages have an `<Extension>` element for services which need more parameters. The `<Extension>` element
691 SHOULD NOT be used in a message, unless its content and related processing rules have been specified for the
692 service. If the receiving party does not support the use of the `<Extension>` element, it MUST ignore it.

693 3.7. General error handling

694 This subchapter defines processing rules for some general error cases.

695 A WSP may not support all different type of requests:

- 696 • A WSP may not support e.g. modifications at all. In the cases, when a WSP receives a `[RequestElement]`,
697 which it does not support, the processing fails and the second level status code `ActionNotSupported` SHOULD
698 be returned in addition to the top level status code. A WSP MAY also register a relevant discovery option
699 keyword to indicate that it does not support certain type of requests although they are available based on the
700 specification for the service a WSP is hosting. Following discovery option keywords are specified for this purpose:
701 `urn:liberty:dst:noQuery`, `urn:liberty:dst:noModify` and `urn:liberty:dst:noSubscribe`
702 `urn:liberty:dst:noQuerySubscriptions`

703 A WSP may encounter problems other than errors in the incoming message:

- 704 • If the processing takes too long (for example some back-end system is not responding fast enough) the second level
705 status code `TimeOut` SHOULD be used to indicate this, when the request is not processed due to a WSP internal
706 time out. The WSP defines how long it tries to process before giving up and returning the `TimeOut` status code.
707 Note that [\[LibertySOAPBinding\]](#) specifies a header block which a WSC may use to define threshold for timeout,
708 but that is different functionality and the processing rules for that are specified in [\[LibertySOAPBinding\]](#).
- 709 • Other error conditions than those listed in this specification and in service specifications may occur. There are two
710 status codes defined for those cases. For cases a WSP (or WSC receiving a notification) can handle normally but
711 for which there is no status code specified, the second level status code `UnspecifiedError` SHOULD be used.
712 For totally unexpected cases the second level status code `UnexpectedError` SHOULD be used.

713 4. Querying Data

714 Two different kind of queries are supported, one for retrieving current data, and another for requesting only changed
715 data. These two different kind of queries can be present together in the same message. The response can contain the
716 data with or without the common technical attributes, depending on the request. Some common attributes are always
717 returned for some elements. When there are multiple elements matching the search criteria, they can be requested in
718 smaller sets and sorted by a defined criteria.

719 4.1. The <Query> Element

720 The <Query> element has two sub-elements. Either the <ResourceID> or the <EncryptedResourceID> element
721 specifies the resource this query is aimed at. The <QueryItem> element specifies what data the requester wants from
722 the resource and how. There can be multiple <QueryItem> elements in one <Query>.

723 The main content the <QueryItem> element has is a <Select> element. The <Select> element specifies the data
724 the query should return and other possible service specific parameters related to the data to be returned. When the
725 select defines that one or more data elements should be returned, then all of these elements and their descendants are
726 returned unless service specific parameters filter out some or all requested data. Also privacy rules may not allow
727 returning some or all of the requested data.

728 The <QueryItem> can also have a <Sort> element. The type and possible content of this element are specified by
729 the services using this feature. The <Sort> element contains the criteria according to which the data in the response
730 should be sorted. E.g. address cards of a contact book could be sorted based on names using either ascending or
731 descending order. As sorting is resource consuming the service specification MUST use sorting very carefully and
732 specify sorting only based on the data and criterias, which are really needed. In many cases sorting on the server side
733 is not needed at all. When sorting is needed, only very limited set of available sorting criterias should be defined.

734 The <QueryItem> can also have a <ChangeFormat> element. The value of this element specifies, in which format
735 the requesting WSC would like to have the data, when querying for changes. Two different formats are defined
736 in this specification. These formats are explained in the processing rules (see [Section 4.3](#)). The schema for the
737 <ChangeFormat> element is:

```
738 <xs:element name="ChangeFormat">  
739 <xs:simpleType>  
740 <xs:restriction base="xs:string">  
741 <xs:enumeration value="ChangedElements"/>  
742 <xs:enumeration value="CurrentElements"/>  
743 </xs:restriction>  
744 </xs:simpleType>  
745 </xs:element>  
746  
747
```

748 The <QueryItem> element can have two attributes qualifying the query in more detail:

749 includeCommonAttributes [Optional]

750 The includeCommonAttributes specifies what kind of response is requested. The default value is *False*,
751 which means that only the data specified in the service definition is returned. If the common attributes
752 specified for container and leaf elements in this document are also needed, then this attribute must be given
753 the value *True*. If the id attribute is used for distinguishing similar elements from one other by the service, it
754 MUST always be returned, even if the includeCommonAttributes is *False*.

755 The xml:lang and script attributes are always returned when they exist.

756 changedSince [Optional]

757 The changedSince attribute should be used when the requester wants to get only the data which has changed
758 since the time specified by this attribute. The changed data can be returned in different ways. A WSC should
759 specify the format it prefers using the element <ChangeFormat>. Please note that use of this changedSince

760 attribute does not require a service to support the common attribute `modificationTime`. The service can
761 keep track of the modification times without providing those times as `modificationTime` attributes for
762 different data elements.

763 In addition to the `id` attribute, the `<QueryItem>` element can have also the `itemID` attribute. This `itemID` attribute
764 is necessary when the `<Query>` element contains multiple `<QueryItem>` elements. The response message can refer
765 to `itemID` attributes of the `<QueryItem>` elements. Also the `<Query>` element can have the `itemID` attribute.
766 `<QueryResponse>` elements in the response message can be mapped to the corresponding `<Query>` elements using
767 this attribute.

768 4.1.1. Pagination

769 When the search criteria defined in the `<Select>` matches multiple elements of same type and name, the WSC may
770 want to have the data in smaller sets, i.e. a smaller number of elements at a time. This is achieved by using the
771 attributes `count`, `offset`, `setID` and `setReq` of the `<QueryItem>` element. The basic attributes are the `count` and
772 the `offset`:

773 `count` [Optional]

774 The `count` attribute defines, how many elements should returned in a response. This is the amount of the
775 elements directly addressed by the `<Select>`, their descendants are automatically included in the response,
776 if not elsewhere otherwise specified.

777 `offset` [Optional]

778 The `offset` attribute specifies, from which element to continue, when querying for more data. The default
779 value is zero, which refers to the first element.

780 Changes may happen while a WSC is requesting the data in smaller sets as this requires multiple `<Query>` messages
781 and so will cause multiple `<QueryResponse>`s. This is not a problem for many services, but with some services
782 this might cause problems as an inconsistent set of data may be returned to the requesting WSC. If supported by the
783 service type and the WSP, a WSC may request that the modifications done by others are not allowed to effect what the
784 requesting WSC gets. In the first `<Query>` of a sequence, the requesting WSC includes the `setReq` attribute with the
785 value `Static`. The query response returns an identification for the set and in the following queries, this is included as
786 the value of the `setID` attribute. At the end the WSC requests that the set is deleted (`setReq="DeleteSet"`) to free
787 the resources on the WSP side.

788 `setID` [Optional]

789 The `setID` attribute contains an identification of a set. This must be used by a WSC, when it wants to make
790 sure that no modifications are done to the set, while it is querying the data from the set.

791 `setReq` [Optional]

792 With the `setReq` attribute a WSC is able to request that a consistent set is created for coming queries (value
793 `Static`) or a set is deleted (`DeleteSet`).

794 A service specification MUST specify, for which elements the pagination is supported. It is not meant to be available
795 for each request, just for a selected type of requests. As the use of the static sets consumes more resources on the
796 server side than normal pagination, the use of static sets must be thought carefully.

797 4.1.2. The schema for the `<Query>` element

```
798 <!-- Querying Data -->
799 <xs:element name="Query" type="QueryType"/>
800 <xs:complexType name="QueryType">
801   <xs:sequence>
802     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
803     <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded"/>
```

```
804     <xs:complexType>
805       <xs:sequence>
806         <xs:annotation>
807           <xs:documentation>
808             NOTE: The below two types (SelectType and SortType) must
809             be defined by the schema that includes this one.
810           </xs:documentation>
811         </xs:annotation>
812         <xs:element name="Select" type="SelectType" minOccurs="0" />
813         <xs:element name="Sort" type="SortType" minOccurs="0" />
814         <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2" />
815       </xs:sequence>
816       <xs:attribute name="id" type="xs:ID" />
817       <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0" />
818       <xs:attribute name="itemID" type="IDType" />
819       <xs:attribute name="changedSince" type="xs:dateTime" />
820       <xs:attribute name="count" type="xs:nonNegativeInteger" />
821       <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0" />
822       <xs:attribute name="setID" type="IDType" />
823       <xs:attribute name="setReq">
824         <xs:simpleType>
825           <xs:restriction base="xs:string">
826             <xs:enumeration value="Static" />
827             <xs:enumeration value="DeleteSet" />
828           </xs:restriction>
829         </xs:simpleType>
830       </xs:attribute>
831     </xs:complexType>
832   </xs:element>
833   <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
834 </xs:sequence>
835 <xs:attribute name="id" type="xs:ID" />
836 <xs:attribute name="itemID" type="IDType" />
837 </xs:complexType>
838
839
```

840 4.2. The <QueryResponse> Element

841 In addition to different identifiers the <QueryResponse> can contain three different things: requested data elements
842 with some parameters, a status code and a time stamp.

843 The requested data is encapsulated inside <Data> elements. One <Data> element contains data requested by one
844 <QueryItem> element. If there were multiple <QueryItem> elements in the <Query>, the <Data> elements are
845 linked to their corresponding <QueryItem> elements using the itemIDRef attributes.

846 If a WSC requested sorting, but a WSP does not support the requested type of sorting or sorting in general, a WSP
847 SHOULD return the data unsorted, but then it MUST indicate this by including the attribute notSorted within the
848 <Data> element carrying the unsorted data. The notSorted attribute may have either the value Now, when the
849 requested sorting is not supported, but sorting in general is, or Never, when the sorting is not supported at all.

850 If a WSC was querying for changes, the <Data> element may contain the attribute changeFormat to indicate in
851 which format the changes are returned. The schema for the changeFormat attribute is following:

```
852   <xs:attribute name="changeFormat">
853     <xs:simpleType>
854       <xs:restriction base="xs:string">
855         <xs:enumeration value="ChangedElements" />
856         <xs:enumeration value="CurrentElements" />
857         <xs:enumeration value="All" />
858       </xs:restriction>
859     </xs:simpleType>
860   </xs:attribute>
```

861
862

863 The <Data> element contains also attributes `nextOffset` and `remaining`, when a smaller set of the data instead
864 all the data was requested using the `count` and the `offset` attributes in the request. The `nextOffset` attribute in
865 a response is the offset of the first item not included in the response. So the value of the `nextOffset` attribute in
866 a response can be used directly for the `offset` attribute in the next request, when the data is fetched sequentially.
867 The `remaining` attribute defines, how many items there are after the last item included in the response. The `setID`
868 attribute is also included, when a static set is accessed.

869 If there were multiple <Query> elements in the request message, the <QueryResponse> elements are linked to
870 corresponding <Query> elements with `itemIDRef` attributes.

871 The schema for <QueryResponse> is below:

```
872 <xs:element name="QueryResponse" type="QueryResponseType" />
873 <xs:complexType name="QueryResponseType">
874   <xs:sequence>
875     <xs:element ref="Status" />
876     <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
877       <xs:complexType>
878         <xs:sequence>
879           <xs:any minOccurs="0" maxOccurs="unbounded" />
880         </xs:sequence>
881         <xs:attribute name="id" type="xs:ID" />
882         <xs:attribute name="itemIDRef" type="IDReferenceType" />
883         <xs:attribute name="notSorted">
884           <xs:simpleType>
885             <xs:restriction base="xs:string">
886               <xs:enumeration value="Now" />
887               <xs:enumeration value="Never" />
888             </xs:restriction>
889           </xs:simpleType>
890         </xs:attribute>
891         <xs:attribute ref="changeFormat" />
892         <xs:attribute name="remaining" type="xs:integer" />
893         <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0" />
894         <xs:attribute name="setID" type="IDType" />
895       </xs:complexType>
896     </xs:element>
897     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
898   </xs:sequence>
899   <xs:attribute name="id" type="xs:ID" />
900   <xs:attribute name="itemIDRef" type="IDReferenceType" />
901   <xs:attribute name="timeStamp" type="xs:dateTime" />
902 </xs:complexType>
903
904
```

905 4.3. Processing Rules

906 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

907 One <Query> element can contain multiple <QueryItem> elements. The following rules specify how those must be
908 supported and handled:

- 909 • A WSP MUST support one <QueryItem> element inside a <Query> and SHOULD support multiple. If a WSP
910 supports only one <QueryItem> element inside a <Query> and the <Query> contains multiple <QueryItem>
911 elements, the processing of the whole <Query> MUST fail and a status code indicating failure MUST be returned
912 in the response. A more detailed status code with the value `NoMultipleAllowed` SHOULD be used in addition

- 913 to the top level status code. If a WSP supports multiple `<QueryItem>` elements inside a `<Query>`, it MAY register
914 the `urn:liberty:dst:multipleQueryItems` discovery option keyword.
- 915 • If the `<Query>` contains multiple `<QueryItem>` elements, the WSP MUST add `itemID` attributes to each
916 `<QueryItem>` element. The WSP MUST link the `<Data>` elements to corresponding `<QueryItem>` elements
917 using the `itemIDRef` attributes, if there were `itemID` attributes in the `<QueryItem>` elements and there were
918 multiple `<QueryItem>` elements in the `<Query>`. The `itemIDRef` attribute in a `<Data>` element MUST have
919 the same value as the `itemID` attribute in the corresponding `<QueryItem>` element.
 - 920 • If processing of a `<QueryItem>` fails, any remaining unprocessed `<QueryItem>` elements SHOULD NOT be
921 processed. The data for the already processed `<QueryItem>` elements SHOULD be returned in the response
922 message and the status code MUST indicate the failure to completely process the whole `<Query>`. A more detailed
923 status SHOULD be used in addition to the top level status code to indicate the reason for failing to process the first
924 failed `<QueryItem>`.
- 925 The following rules specify how the `<Select>` element should be processed and interpreted:
- 926 • If there is no `changedSince` attribute in the `<QueryItem>` element and the `<Select>` requests valid data
927 element(s), but there are no values, the WSP MUST NOT return any `<Data>` element for that `<QueryItem>`
928 unless a WSP is requesting pagination. In this case a WSP MUST return the `<Data>` element containing the
929 remaining and the `nextOffset` attributes even, when no actual data is returned (see processing rules related to
930 pagination later on).
 - 931 • If the `<Select>` requests multiple data elements, the WSP MUST return all of those data elements inside the
932 `<Data>` element corresponding to the containing `<QueryItem>`.
- 933 A WSP may request that the data returned is sorted.
- 934 • When the `<Sort>` element is included in a `<QueryItem>` element, the data returned inside a `<Data>` element
935 SHOULD be sorted according to the criteria given in the `<Sort>` element. If a WSP doesn't support sorting,
936 it SHOULD return the requested data unsorted. When the data is returned unsorted, the `notSorted` attribute
937 MUST be used in the `<Data>` element containing the unsorted data. A WSP MAY also choose to fail to process
938 the `<QueryItem>`, if it does not support sorting. In that case the second level status code `SortNotSupported`
939 SHOULD be used in addition to the top level status code. A WSP may also register discovery option keyword
940 `urn:liberty:dst:noSorting`, if the sorting has been specified for the service type, but the WSP doesn't
941 support it.
 - 942 • If the content of the `<Sort>` element is not according to service specifications, a WSP SHOULD return the
943 requested data unsorted. When the data is returned unsorted, the `notSorted` attribute MUST be used in the
944 `<Data>` element containing the unsorted data and the second level status code `InvalidSort` SHOULD also be
945 used. A WSP MAY also choose to fail to process the `<QueryItem>`, if the content of the `<Sort>` element is not
946 according to service specifications. In this kind of a case the second level status code `InvalidSort` SHOULD be
947 used in addition to the top level status code. If the the content of the `<Sort>` element is valid, but a WSP does not
948 support the requested type of sorting, it SHOULD return the requested data unsorted. When the data is returned
949 unsorted, the `notSorted` attribute MUST be used in the `<Data>` element containing the unsorted data. A WSP
950 MAY also choose to fail to process of the `<QueryItem>`, if it does not support the requested type of sorting. It
951 SHOULD use the second level status code `RequestedSortingNotSupported` in addition to the top level status
952 code.
 - 953 • When the `notSorted` attribute is used, it MUST have the value `Now`, when a WSP supports sorting, but not the
954 requested type or the content of the `<Sort>` element was invalid. The `notSorted` attribute MUST have the value
955 `Never`, when a WSP does not support sorting at all.

- 956 A WSC may want to receive the data in smaller sets instead of getting all the data at once, when there can be many
957 elements with the same name. A WSC indicates this using either or both of the attributes `count` and `offset` in a
958 `<QueryItem>` element, when the `<Select>` addresses a set of elements all having the same name. The number of
959 elements inside this set may be restricted further by other parameters. Also access rights and policies may reduce the
960 set of elements a WSC is allowed to get.
- 961 • A WSP MUST always follow the same ordering procedure, when the `<Select>` and `<Sort>` elements have the
962 same values and either or both of attributes `count` and `offset` are used in the `<QueryItem>` element. This is
963 needed to make sure e.g. that a WSC really gets the next ten items, when asking for them, and not e.g. five of the
964 previously returned items with five new items.
 - 965 • When either or both of the attributes `count` and `offset` is used in a `<QueryItem>` element and a WSP doesn't
966 support pagination, the processing of whole `<QueryItem>` element MUST fail and the second level status code
967 `PaginationNotSupported` SHOULD be used in addition to the top level status code. A WSP may support
968 pagination, but not for the requested elements. In such a case the processing of whole `<QueryItem>` element
969 MUST fail and the second level status code `RequestedPaginationNotSupported` SHOULD be used in
970 addition to the top level status code. If a WSP doesn't support pagination at all, it MAY register the discovery
971 option keyword `urn:liberty:dst:noPagination` to indicate this.
 - 972 • When the `count` attribute is included in a `<QueryItem>` element, the corresponding `<Data>` element in the
973 `<QueryResponse>` MUST NOT contain more elements addressed with the value of the `<Select>` element than
974 specified by the `count` attribute. A WSP MAY return a smaller number of elements of the same name that
975 requested by a WSC. If the `count` attribute has the value zero, the WSP MUST NOT return any data elements
976 inside the `<Data>` element. This `count="0"` may be used for querying the number of remaining elements starting
977 from the specified offset, e.g. from offset zero, i.e. the total number of the elements addressed by the `<Select>`
978 element. When the `count` attribute is not used in a `<QueryItem>` element, it means that the WSC requests for
979 all data specified by other parameters like the `<Select>` element starting from the specified offset. As the default
980 value for the `offset` attribute is zero, the case when neither of the attributes `offset` or `count` is not present
981 reduces to a normal query.
 - 982 • When pagination is requested by a WSC, the elements inside a `<Data>` element MUST be in the ascending order
983 of their offsets. The first element MUST have the offset specified by the `offset` attribute in the `<QueryItem>`
984 element. The `<Data>` element MUST have both attributes `nextOffset` and `remaining`. The `nextOffset`
985 attribute MUST have the offset of the first element not returned in the response. The value of the `remaining`
986 attribute MUST define how many elements there are left starting from the value of the `nextOffset`, if a WSP
987 knows that (e.g. that information might not be available from a backend system). If WSP does not know the
988 exact value, it MUST use the value `-1` for the `remaining` attribute until it knows the value or there is no data left
989 (`remaining="0"`). When `remaining="-1"`, a WSC must make new requests until `remaining="0"`, if it wants
990 to get all the data.
 - 991 • Usually, when there is no data matching the different query parameters, no `<Data>` element is returned in a
992 `<QueryResponse>`. When either or both of the `count` and `offset` attributes are used, the `<Data>` element
993 MUST be returned, even, when no data is returned (e.g. no data available or `count="0"` used to get the number
994 of data items). This is required so that a WSP can return the `remaining` and the `nextOffset` attributes to the
995 requesting WSC.
 - 996 • When the `setReq` attribute is included in a `<QueryItem>` element and has the value `Static`, the WSP SHOULD
997 return the `setID` attribute to the requesting WSC and process `<QueryItem>` elements later having this `setID`
998 based on the data the WSP has at the time, when the value for the `setID` was created. If a WSP receives a
999 `<QueryItem>` element having the `setReq` attribute and does not support static sets for the requested data or not at
1000 all, the processing of the `<QueryItem>` element MUST fail and a second level status code `StaticNotSupported`
1001 SHOULD be used in addition to the top level status code. If a WSP doesn't support static sets at all, it MAY register
1002 the discovery option keyword `urn:liberty:dst:noStatic`.

- 1003 • When the `setID` attribute is included in a request, the following parameters MUST NOT be used in a
1004 `<QueryItem>` element: the `<Select>` element, the `<Sort>` element, the `changedSince` attribute or the
1005 `includeCommonAttributes` attribute. The requests are made from an earlier defined static set and the `count`
1006 and the `offset` attributes are used to define, what is requested from that set. If any of the mentioned parameters
1007 is present, when the `setID` attribute is used, it is unclear what a WSC wants and the processing of the whole
1008 `<QueryItem>` MUST fail and a second level status code `SetOrNewQuery` SHOULD be used in addition to the
1009 top level status code.
- 1010 • When the `setID` attribute is included in a `<QueryItem>` element and has a valid value, the `<Data>` element in
1011 the response MUST always have the `setID` attribute.
- 1012 • When a static set is created, the requesting WSC MUST query all the data it needs from this set as soon as possible
1013 and delete the static set immediately after this using `setReq="DeleteSet"`. A WSP MAY also delete the static
1014 set, even if a WSC hasn't yet requested the deletion of the static set. If a WSC tries to make a request to a
1015 non-existing static set, the processing of the whole `<QueryItem>` MUST fail and the second level status code
1016 `InvalidSetID` SHOULD be used in addition to the top level status code.
- 1017 • The `setReq="Static"` and the `setID` attribute MUST NOT be used simultaneously in a `<QueryItem>` element.
1018 If they are used, the WSP MUST ignore the `setReq="Static"` and process the `<QueryItem>` element like the
1019 `setReq` attribute would not be present.
- 1020 • If the `setReq` attribute has some other value than `Static` or `DeleteSet`, the processing of the whole
1021 `<QueryItem>` element must fail and a second level status code `InvalidSetReq` SHOULD be used in addition
1022 to the top level status code.
- 1023 Even when the requested data exists, it should be noted that access and privacy policies specified by the resource owner
1024 may cause the request to result in data not being returned to the requester.
- 1025 • When a WSP processes a `<QueryItem>`, it MUST check whether the resource owner (the Principal, for example)
1026 has given consent to return the requested information. To be able to check WSC specific access rights, the WSP
1027 MUST authenticate the WSC (see [\[LibertySecMech\]](#) and [\[LibertyMetadata\]](#)). The WSP MUST also check that
1028 any usage directive given in the request is acceptable based on the usage directives defined by the resource owner
1029 (see [\[LibertySOAPBinding\]](#)). If either check fails for any piece of the requested data, the WSP MUST NOT return
1030 that piece of data. Note that there can be consent for returning some data element, but not its attributes. E.g. a
1031 resource owner might not want to release the `modifier` attribute, if she does not want to reveal information about
1032 which services she uses. The data for which there is no consent from the resource owner MUST be handled as
1033 if there was no data. The WSP MAY try to get consent from the resource owner while processing the request,
1034 e.g. by using an interaction service (see [\[LibertyInteract\]](#)). A WSP might check the access rights and policies
1035 in usage directives at a higher level, before getting to DST processing and MAY, in this case, just return an ID-
1036 Fault Message [\[LibertySOAPBinding\]](#) without processing the `<Query>` element at all, if the requesting WSC is
1037 not allowed to access data.
- 1038 It is possible to query changes since a specified time using the `changedSince` attribute. The following rules specify
1039 how this works:
- 1040 • If the `<QueryItem>` element contains the `changedSince` attribute, the WSP SHOULD return only those elements
1041 addressed by the `<Select>` which have been modified since the time specified in the `changedSince` attribute.
1042 There are two different formats, in which the changed data can be returned. A WSC SHOULD indicate using
1043 the `<ChangeFormat>` element the format it prefers and also, if it understands the other format. The two formats
1044 are `ChangedElements` and `CurrentElements`. If a service specification doesn't specify anything else the value
1045 `ChangedElements` MUST be used as a default value as it is compatible with the format used in the version 1.0
1046 of the Data Services Template.

- 1047 • A WSP MUST ignore the `<ChangeFormat>` element, if the `changedSince` attribute is not used in the same
1048 `<QueryItem>` element. A WSP MUST NOT use a format, which a WSC does not understand. Note that format
1049 `ChangedElements`, has the format `All` as a fallback solution, when a WSP doesn't have all the needed change
1050 history information. Also if a WSP doesn't support requesting only changed data, it returns all data.
- 1051 • A `<QueryItem>` element MAY contain two `<ChangeFormat>` element with different values. A WSP SHOULD
1052 use the format specified by the first `<ChangeFormat>` element, but, if it does not support that format, it MAY use
1053 the format specified by the second `<ChangeFormat>` element.
- 1054 • If a WSP does not support the format a WSC is requesting to be used, the processing of the `<QueryItem>` MUST
1055 fail and the second level status code `FormatNotSupported` SHOULD be used in addition to the top level status
1056 code.
- 1057 • If a WSC requests the `ChangedElements` format and a WSP supports it, the WSP SHOULD return only the
1058 changed information. if some element has been deleted, a WSP SHOULD return an empty element to indicate the
1059 deletion (`<ElementName/>`). The only allowed exception to this is that the WSP does not have enough history
1060 information available to be able to return only the changed information. In that case it MUST use format `All` and
1061 return all current elements with their values even if those have not changed since the specified time.
- 1062 • If a WSC requests the `CurrentElements` format and a WSP supports it, the WSP SHOULD return only the
1063 currently existing elements. It SHOULD return an empty element, if an element has not changed to indicate that
1064 no change has happened (`<ElementName/>`).
- 1065 • Note: as empty elements are used to indicate either deleted or not changed elements depending on the used format,
1066 the formats `CurrentElements` and `ChangedElements` do not work well, if the data hosted by a service may
1067 contain empty elements. In those cases a service should either use only format `All` or always have some attribute(s)
1068 for the otherwise empty elements.
- 1069 • If a WSC has used the `<ChangeFormat>` element in a request, a WSP MUST use the `changeFormat` attribute
1070 in the response to indicate, which format is used. A WSP MUST not use the `changeFormat` attribute in
1071 a response, if the `<ChangeFormat>` element was not used in the corresponding request so the processing stays
1072 version 1.0 compatible, when the `<ChangeFormat>` element is not used.
- 1073 • If there can be multiple elements with same name, the `id` attribute or some other attribute used to distinguish
1074 the elements from each other MUST be included (e.g. in case of an ID-SIS Personal Profile service the
1075 following empty element could be returned `<AddressCard id="tr7632q"/>` to indicate a deleted or not
1076 changed `<AddressCard>` depending on the used format). If the value of the `id` attribute or some other attribute
1077 used for distinguishing elements with same name is changed, the WSP MUST consider this as a case, in which the
1078 element with the original value of the distinguishing attribute is deleted and a new one with the new value of the
1079 distinguishing attribute is created. To avoid this, a WSP MAY refuse to accept modifications of a distinguishing
1080 attribute and MAY require that an explicit deletion of the element is done and a new one created.
- 1081 • If the elements addressed by the `<Select>` have some values, but there has been no changes since the time
1082 specified in the `changedSince` attribute, the WSP MUST return empty `<Data>` element (`<Data/>`), when it
1083 returns the changes properly. This empty `<Data>` element indicates that no changes have occurred. There might
1084 be cases in which the WSP is not able to return changes properly, see later processing rules. Please note that in
1085 cases that have no values, no `<Data>` element is returned to indicate this. So empty `<Data>` element has different
1086 semantics than missing `<Data>` element.

- 1087 • If the `<QueryItem>` element contains the `changedSince` attribute and a WSP is not keeping track of modification
1088 times, it SHOULD process the `<QueryItem>` element as there would be no `changedSince` attribute, and indicate
1089 this in the response using the second level status code `ChangedSinceReturnsAll`. This is not considered a
1090 failure and the rest of the `<QueryItem>` elements MUST be processed. Also it might be that a WSP does not
1091 have a full change history and so for some queries, it is not possible to find out, which changes occurred after
1092 the specified time. As processing with access rights and policy in place might be quite complex, a WSP might
1093 sometimes process the query for changes properly and sometime process it as if there were no `changedSince`
1094 attribute. In those cases, when a WSP returns all current values, it SHOULD indicate this with the second level
1095 status code `AllReturned` and, if the `<ChangeFormat>` element was used in the request, the `changeFormat`
1096 attribute with the value `All` SHOULD be used. This is also not considered a failure and the rest of the
1097 `<QueryItem>` elements MUST be processed. Please note that the status code `AllReturned` differs from the
1098 status code `ChangedSinceReturnsAll`, as `ChangedSinceReturnsAll` means that the WSP never processes
1099 the `changedSince` attribute properly. A WSP MUST use either `AllReturned` or `ChangedSinceReturnsAll`
1100 as the second level status code, when it returns data, but does not process the `changedSince` attribute properly, i.e.
1101 returns only the changes. If a WSP will not process the `<QueryItem>` elements with a `changedSince` attribute
1102 at all, it MUST indicate this with top level status code `Failed` and SHOULD also return a second level status code
1103 of `ChangeHistoryNotSupported` in the response. In this case a WSP MUST NOT return any `<Data>` element
1104 for the `<QueryItem>` element containing the `changedSince` attribute. If a WSP processes the `changedSince`
1105 attribute, it MUST also support the `notChangedSince` attribute for `<Modification>` element and MAY register
1106 the `urn:liberty:dst:changeHistorySupported` discovery option keyword. Please note that still in some
1107 cases a WSP MAY return `AllReturned`.
- 1108 • Access rights and policies in place may affect how the queries for changes can work as they affect which elements
1109 and attributes a WSC is allowed to see. If a WSC was originally allowed to get the requested data, but is no longer
1110 after some change in access policies, then from its point of view that data is deleted and that should be taken into
1111 account in the response. If the WSP notices that access rights have changed, and the current rights do not allow
1112 access, it MUST return all data except the data for which the access rights were revoked, and use the second level
1113 status code `AllReturned` and, if the `<ChangeFormat>` element was used in the request, the `changeFormat`
1114 attribute with the value `All` SHOULD be used. The WSP MUST NOT return empty elements for the data for
1115 which access rights were changed even if the format `ChangedElement` was requested, as this might reveal the
1116 fact that this specific data has at least existed at the service in some point of time. Please note that it might be the
1117 case that the data was added after the WSCs access rights were revoked and the WSC was never supposed to be
1118 aware of the existence of that data. If the WSP notices that the access rights are changed and the current rights do
1119 allow access, it MUST consider the data for which the access rights are changed, as if it were just created.
- 1120 • Both the WSC and WSP may have policies specified by the Principal for control of their data. Only by
1121 comparing policy statements made by the WSC (via `<UsageDirective>` elements (see [LibertySOAPBinding](#)))
1122 with policies maintained on behalf of the Principal by the WSP it is possible to fully determine the effects of
1123 interaction between these sets of policies. As it might be too expensive to search for policies the WSC promised
1124 to honour, when it made the original request, and this information might not even be available, the WSP might be
1125 only capable of making the decision based on the policy changes made by the Principal. If some data is prevented
1126 from being returned to the WSC due to conflicts in policies and the WSP notices that the Principal's policies have
1127 changed, it MUST return all data except that for which the Principal's policy has denied access against the current
1128 policy of a requesting WSC, and use the second level status code `AllReturned` to indicate that the WSC must
1129 check the response carefully to find out what has changed. Also if the `<ChangeFormat>` element was used in
1130 the request, the `changeFormat` attribute with the value `All` SHOULD be used. The WSP MUST NOT return
1131 empty elements for the data for which the Principal's policy was changed even if the format `ChangedElements`
1132 was requested, as this might reveal the fact that this specific data was exposed by the service at some point in time.
1133 Please note that it might be the case that that data has been added after the policies were changed and the requesting
1134 WSC was never supposed to be aware of that data, unless it changed the policy it promises to honour. If the WSP
1135 notices that the Principal's policy has changed and the current policy does allow access, it MUST consider the data
1136 for which the policy is changed as if it had been just created. If a WSC changes the policy it promises to honour,
1137 it SHOULD make a new query without a `changedSince` attribute.

1138 • As mentioned earlier the WSP might in some cases return all the current data the <Select> points to, and not
1139 just the changes using specified format, even when the `changedSince` attribute is present. So the WSC MUST
1140 compare the returned data to previous data it had queried earlier to find out what really has changed. Please note
1141 that this MUST be done even when the WSP has processed the `changedSince` correctly, because some values
1142 might have been changed back and forth and now they have same values that they used to have earlier, despite the
1143 most current previous values being different.

1144 The common attributes are not always returned. A WSC may indicate with the `includeCommonAttributes`
1145 attribute, whether it wants to have the common attributes or not.

1146 • If the `includeCommonAttributes` is set to *True*, the common attributes specified by attribute
1147 groups `commonAttributes` and `leafAttributes` MUST be included in the response, if their val-
1148 ues are specified for the requested data elements. The ACC attributes MAY be left out, if the value is
1149 `urn:liberty:dst:acc:unknown`.

1150 • If the `id` attribute is used for distinguishing similar elements from each other by the service, it MUST be returned,
1151 even if the `includeCommonAttributes` is false. Also, when either or both of the attributes `xml:lang` and
1152 `script` are present, they MUST be returned, even if the `includeCommonAttributes` is false

1153 4.4. Examples

1154 Please note that as the first examples are based on the [\[LibertyIDPP\]](#) all elements defined in this specifications are also
1155 in the namespace defined in [\[LibertyIDPP\]](#).

1156 The following query example requests the common name and home address of a Principal:

```
1157  
1158 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">  
1159   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>  
1160   <pp:QueryItem itemID="name">  
1161     <pp:Select>/pp:PP/pp:CommonName</pp:Select>  
1162   </pp:QueryItem>  
1163   <pp:QueryItem itemID="home">  
1164     <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType="urn:liberty:id-sis-pp:addrType:  
1165 home"]</pp:Select>  
1166   </pp:QueryItem>  
1167 </pp:Query>  
1168
```

1169 This query may generate the following response:

```
1170  
1171 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08">  
1172   <pp:Status code="OK"/>  
1173   <pp:Data itemIDRef="name">  
1174     <pp:CommonName>  
1175       <pp:CN>Zita Lopes</pp:CN>  
1176       <pp:AnalyzedName nameScheme="firstlast">  
1177         <pp:FN>Zita</pp:FN>  
1178         <pp:SN>Lopes</pp:SN>  
1179         <pp:PersonalTitle>Dr.</pp:PersonalTitle>  
1180       </pp:AnalyzedName>  
1181       <pp:AltCN>Maria Lopes</pp:AltCN>  
1182       <pp:AltCN>Zita Ma Lopes</pp:AltCN>  
1183     </pp:CommonName>  
1184   </pp:Data>  
1185   <pp:Data itemIDRef="home">  
1186     <pp:AddressCard id='9812'>  
1187       <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
```

```
1188     <pp:Address>
1189         <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way North</pp:PostalAddress>
1190         <pp:PostalCode>98503-2341</pp:PostalCode>
1191         <pp:L>Olympia</pp:L>
1192         <pp:ST>wa</pp:ST>
1193         <pp:C>us</pp:C>
1194     </pp:Address>
1195 </pp:AddressCard>
1196 </pp>Data>
1197 </pp:QueryResponse>
1198
```

1199 If there was no user consent for the release of the <pp:CommonName> or for the whole <pp:AddressCard> with
1200 pp:AddressType='urn:liberty:id-sis-pp:addrType:home', apart from the country information, then the
1201 response is as follows (including a timestamp, as this service supports change history).

```
1202
1203 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1204 timeStamp="2003-02-28T12:10:12Z">
1205     <pp:Status code="OK"/>
1206     <pp>Data itemIDRef="home">
1207         <pp:AddressCard id='9812'>
1208             <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1209             <pp:Address>
1210                 <pp:C>us</pp:C>
1211             </pp:Address>
1212         </pp:AddressCard>
1213     </pp>Data>
1214 </pp:QueryResponse>
1215
```

1216 If there was no <pp:CommonName> and no <pp:AddressCard> with pp:AddressType =
1217 'urn:liberty:id-sis-pp:addrType:home', then the response is:

```
1218
1219
1220 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1221 timeStamp="2003-02-28T12:10:12Z">
1222     <pp:Status code="OK"/>
1223 </pp:QueryResponse>
1224
1225
```

1226 The following request queries the fiscal identification number of the Principal with the common attributes:

```
1227
1228
1229 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1230     <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1231     <pp:QueryItem includeCommonAttributes="True">
1232         <pp>Select>/pp:PP/pp:LegalIdentity/pp:VAT</pp>Select>
1233     </pp:QueryItem>
1234 </pp:Query>
1235
1236
```

1237 This query may generate the following response:

```
1238
1239
1240 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1241 id="12345" timeStamp="2003-05-28T23:10:12Z">
1242     <pp:Status code="OK"/>
1243     <pp>Data>
1244         <pp:VAT modifier="http://www.accountingservices.com"
1245             modificationTime="2003-04-25T15:42:11Z">
```

```
1246     attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">
1247     <pp:IDValue modifier="http://www.accountingservices.com"
1248         modificationTime="2003-04-25T15:42:11Z"
1249         attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">50267
1250 7123</IDValue>
1251     <pp:IDType modifier="http://www.accountingservices.com"
1252         modificationTime="2003-03-12T09:12:09Z"
1253         attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">urn:1
1254 iberty:altIDType:itcif</IDType>
1255     </pp:VAT>
1256 </pp:Data>
1257 </pp:QueryResponse>
1258 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
1259     . . .
1260 </ds:Signature>
1261
1262
```

1263 The following request queries for address information which has been changed since 12:10:12 28 February 2003 UTC:

```
1264
1265
1266 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1267     <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1268     <pp:QueryItem changedSince="2003-02-28T12:10:12Z">
1269         <pp>Select>/pp:PP/pp:AddressCard</pp>Select>
1270     </pp:QueryItem>
1271 </pp:Query>
1272
1273
```

1274 This query can generate following response:

```
1275
1276
1277 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1278     timeStamp="2003-05-30T16:10:12Z">
1279     <pp:Status code="OK"/>
1280     <pp:Data>
1281         <pp:AddressCard id='9812'>
1282             <pp:Address>
1283                 <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
1284             </pp:Address>
1285         </pp:AddressCard>
1286         <pp:AddressCard id='wlq2' />
1287     </pp:Data>
1288 </pp:QueryResponse>
1289
1290
```

1291 Please note that only the changed information inside the <pp:AddressCard> is returned. The response shows that
1292 after the specified time, there was also another <pp:AddressCard> present, but that has been deleted. As there can
1293 be many <pp:AddressCard> elements, the id attribute is returned to distinguish distinct elements.

1294 If there have been no changes since the specified time, then the response is just:

```
1295
1296 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1297     timeStamp="2003-05-30T16:10:12Z">
1298     <pp:Status code="OK"/>
1299     <pp:Data/>
1300 </pp:QueryResponse>
1301
```

1302 If the same request for changed addresses is made including the <pp:ChangeFormat> element:

```
1303
1304
1305 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1306   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1307   <pp:QueryItem changedSince="2003-02-28T12:10:12Z">
1308     <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
1309     <pp:ChangeFormat>CurrentElements</pp:ChangeFormat>
1310   </pp:QueryItem>
1311 </pp:Query>
1312
1313
```

1314 All the current elements are returned in the response:

```
1315
1316
1317 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1318 timeStamp="2003-05-30T16:10:12Z">
1319   <pp:Status code="OK"/>
1320   <pp:Data changeFormat="CurrentElements">
1321     <pp:AddressCard id='9812'>
1322       <pp:Address>
1323         <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
1324         <pp:PostalCode/>
1325         <pp:L/>
1326         <pp:ST/>
1327         <pp:C/>
1328       </pp:Address>
1329     </pp:AddressCard>
1330   </pp:Data>
1331 </pp:QueryResponse>
1332
1333
```

1334 Please note that now all the current elements inside the `<pp:AddressCard>` are returned. The deleted
1335 `<pp:AddressCard>` is not shown at all and for the elements, which have not changed only empty elements
1336 are returned.

1337 If a WSP does not support change history, then the response could be:

```
1338
1339 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1340 timeStamp="2003-05-30T16:10:12Z">
1341   <pp:Status code="OK">
1342     <Status code="ChangeSinceReturnsAll"/>
1343   </pp:Status>
1344   <pp:Data changeFormat="All">
1345     <pp:AddressCard id='9812'>
1346       <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1347       <pp:Address>
1348         <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
1349         <pp:PostalCode>98503-2341</pp:PostalCode>
1350         <pp:L>Olympia</pp:L>
1351         <pp:ST>wa</pp:ST>
1352         <pp:C>us</pp:C>
1353       </pp:Address>
1354     </pp:AddressCard>
1355   </pp:Data>
1356 </pp:QueryResponse>
1357
```

1358 The rest of the examples are related to pagination and sorting based on fictional address service, so all the DST
1359 elements in the namespace of that fictional address service.

1360 Parameters <Select> and <Sort> and returned <Data> elements do not have valid contents in the examples as the
1361 main point is to show the principle how pagination works and the use of the pagination related attributes

1362 A Resource contains 40 address cards and the WSC A wants to list those ordered by the City and 10 at the time. Due
1363 to access rights and policies the WSC A is allowed to get only 30 of those AddressCards. The WSC A makes the first
1364 query:

```
1365  
1366 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1367   <ads:ResourceID>http://provider.example.com/ohj243hj24</ads:ResourceID>  
1368   <ads:QueryItem count="10">  
1369     <ads:Select>Pointing to the AddressCards</ads:Select>  
1370     <ads:Sort>Requesting sorting by the City</ads:Sort>  
1371   </ads:QueryItem>  
1372 </ads:Query>  
1373  
1374
```

1375 and gets back the first ten address cards ordered by the City:

```
1376  
1377 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1378 timeStamp="2004-03-23T03:40:00Z">  
1379   <ads:Status code="OK"/>  
1380   <ads:Data remaining="20" nextOffset="10">first ten address cards</ads:Data>  
1381 </ads:QueryResponse>  
1382
```

1383 Then it queries the next ten starting from offset 10:

```
1384  
1385 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1386   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>  
1387   <ads:QueryItem count="10" offset="10">  
1388     <ads:Select>Pointing to the AddressCards</ads:Select>  
1389     <ads:Sort>Requesting sorting by the City</ads:Sort>  
1390   </ads:QueryItem>  
1391 </ads:Query>  
1392
```

1393 and gets those

```
1394  
1395 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1396 timeStamp="2004-03-23T03:40:20Z">  
1397   <ads:Status code="OK"/>  
1398   <ads:Data remaining="10" nextOffset="20">next ten address cards</ads:Data>  
1399 </ads:QueryResponse>  
1400
```

1401 After this the WSC B adds one more address card to the resource. The WSC A is allowed to get this address card, but
1402 when sorting based on the City, this new card has the offset 15. When the WSC A fetches the next ten address cards:

```
1403  
1404 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1405   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>  
1406   <ads:QueryItem count="10" offset="20">  
1407     <ads:Select>Pointing to the AddressCards</ads:Select>  
1408     <ads:Sort>Requesting sorting by the City</ads:Sort>  
1409   </ads:QueryItem>  
1410 </ads:Query>  
1411
```

1412 It gets ten address cards, but it has already received the first address card already in the previous response.

```
1413
1414 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1415 timeStamp="2004-03-23T03:41:00Z">
1416   <ads:Status code="OK"/>
1417   <ads:Data remaining="1" nextOffset="30">next ten address cards</ads:Data>
1418 </ads:QueryResponse>
1419
```

1420 Finally the WSC A fetches the last address card.

```
1421
1422 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1423   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
1424   <ads:QueryItem count="1" offset="30">
1425     <ads:Select>Pointing to the AddressCards</ads:Select>
1426     <ads:Sort>Requesting sorting by the City</ads:Sort>
1427   </ads:QueryItem>
1428 </ads:Query>
1429
```

1430 and gets the 31st address card from offset 30.

```
1431
1432 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1433 timeStamp="2004-03-23T03:41:17Z">
1434   <ads:Status code="OK"/>
1435   <ads:Data remaining="0" nextOffset="31">the last address card</ads:Data>
1436 </ads:QueryResponse>
1437
```

1438 So the WSC A didn't get this new address card added by the WSC B and got one card twice.

1439 In an alternative scenario, if supported by the WSP, the WSC A requests a static set to be created so that simultaneous
1440 modifications can not affect the results the WSC A gets. The initial request includes the `setReq` attribute:

```
1441
1442 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1443   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
1444   <ads:QueryItem count="10" setReq="Static">
1445     <ads:Select>Pointing to the AddressCards</ads:Select>
1446     <ads:Sort>Requesting sorting by the City</ads:Sort>
1447   </ads:QueryItem>
1448 </ads:Query>
1449
```

1450 In the response the first ten address cards are returned together with a handle to this static set (the attribute `setID`).

```
1451
1452 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1453 timeStamp="2004-03-23T03:40:00Z">
1454   <ads:Status code="OK"/>
1455   <ads:Data remaining="20" nextOffset="10" setID="gfkjds98">first ten address cards</ads:Data>
1456 </ads:QueryResponse>
1457
```

1458 In the next query the WSC A queries the next ten address card referring to the static set using the `setID` attribute. The
1459 `<Select>` element is not anymore used.

```
1460
1461 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1462   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
1463   <ads:QueryItem count="10" offset="10" setID="gfkjds98"/>
1464 </ads:Query>
1465
```

1466 In the response the next ten address cards are returned and the setID is still returned as always when accessing a static
1467 set.

```
1468  
1469 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1470 timeStamp="2004-03-23T03:40:00Z">  
1471   <ads:Status code="OK"/>  
1472   <ads:Data remaining="10" nextOffset="20" setID="gfkjds98">next ten address cards</ads:Data>  
1473 </ads:QueryResponse>  
1474
```

1475 When the WSC B tries to add a new address card, it doesn't affect the data the WSC A gets, when requesting the next
1476 ten address cards.

```
1477  
1478 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1479   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>  
1480   <ads:QueryItem count="10" offset="20" setID="gfkjds98"/>  
1481 </ads:Query>  
1482
```

1483 So the WSC A gets the last ten address card.

```
1484  
1485 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1486 timeStamp="2004-03-23T03:40:00Z">  
1487   <ads:Status code="OK"/>  
1488   <ads:Data remaining="0" nextOffset="30" setID="gfkjds98">next ten address cards</ads:Data>  
1489 </ads:QueryResponse>  
1490
```

1491 Finally the WSC A deletes the static set. This deletion could have been done together with the previous request, but
1492 the WSC wanted to play safe and delete the static set only after getting all the data it wanted.

```
1493  
1494 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1495   <ads:ResourceID>http://provider.com/ohj243hj24</ResourceID>  
1496   <ads:QueryItem count="0" setID="gfkjds98" setReq="DeleteSet"/>  
1497 </ads:Query>  
1498
```

1499 And the WSP acknowledges the request.

```
1500  
1501 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1502 timeStamp="2004-03-23T03:40:00Z">  
1503   <ads:Status code="OK"/>  
1504 </ads:QueryResponse>  
1505
```

1506 So the addition the WSC B tried to make is not visible in the static set. Either the WSP refused to accept the addition
1507 while WSC A was accessing the data or it created a temporary set for the WSC A to access and the modification by the
1508 WSC B was accepted, but not visible in the temporary static set created for WSC A. In the example above the WSP
1509 created a temporary set and so returned the same time stamp in all responses containing data from that temporary set.

1510 5. Modifying Data

1511 The data stored by a data service can be given initial values, existing values can be replaced with new values and the
 1512 data can also be removed. Usually the Principal can make these modifications directly at the data service using the
 1513 provided user interface, but these modifications may also be made by other service providers. The <Modify> element
 1514 supports all these operations for service providers which want to modify the data store in data services.

1515 5.1. <Modify> element

1516 The <Modify> element has two sub-elements. Either the <ResourceID> or <EncryptedResourceID> element is
 1517 used to identify the resource which is modified by this request. The <Modification> element specifies which data
 1518 elements of the specified resource should be modified and how. There can be multiple <Modification> elements in
 1519 one <Modify>.

1520 The <Select> element inside a <Modification> element specifies the data this modification should affect. If the
 1521 <Select> element is not used, it means that the whole content of the Resource should be modified. In addition to this
 1522 <Select> element the other main part of the <Modification> element is the <NewData> element. The <NewData>
 1523 element defines the new values for the data addressed by the <Select> element. The new values specified inside the
 1524 <NewData> element replace existing data, if the overrideAllowed attribute of the <Modification> element is
 1525 set to *True*. If the <NewData> element does not exist or is empty, it means than the current data values should be
 1526 removed. The default value for the overrideAllowed attribute is *False*, which means that the <Modification>
 1527 is only allowed to add new data, not to remove or replace existing data. The notChangedSince attribute is used
 1528 to handle concurrent updates. When the notChangedSince attribute is present, the modification is allowed to be
 1529 done only if the data to be modified has not changed since the time specified by the value of the notChangedSince
 1530 attribute. The <Modification> element **MUST** also have the itemID attribute, when multiple <Modification>
 1531 elements are included in one <Modify> element and partial failure is allowed so that failed parts can be identified.

1532 In addition to the id attribute, the <Modify> element can have also the itemID attribute. This is necessary when
 1533 the request message has multiple <Modify> elements. The response message can refer to itemID attributes of
 1534 the <Modify> elements and so map <ModifyResponse> elements in the response message to the corresponding
 1535 <Modify> elements.

1536 The schema for <Modify>

```

1537 <!-- Modifying Data -->
1538 <xs:element name="Modify" type="ModifyType"/>
1539 <xs:complexType name="ModifyType">
1540   <xs:sequence>
1541     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
1542     <xs:element name="Modification" maxOccurs="unbounded">
1543       <xs:complexType>
1544         <xs:sequence>
1545           <xs:annotation>
1546             <xs:documentation>
1547               NOTE: The below SelectType must be defined by
1548               the schema that includes this one.
1549             </xs:documentation>
1550           </xs:annotation>
1551           <xs:element name="Select" type="SelectType" minOccurs="0"/>
1552           <xs:element name="NewData" minOccurs="0">
1553             <xs:complexType>
1554               <xs:sequence>
1555                 <xs:any minOccurs="0" maxOccurs="unbounded"/>
1556               </xs:sequence>
1557             </xs:complexType>
1558           </xs:element>
1559         </xs:sequence>
1560         <xs:attribute name="id" type="xs:ID"/>
1561         <xs:attribute name="itemID" type="IDType"/>
1562         <xs:attribute name="notChangedSince" type="xs:dateTime"/>
  
```

```

1563         <xs:attribute name="overrideAllowed" type="xs:boolean" default="0" />
1564     </xs:complexType>
1565 </xs:element>
1566     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1567 </xs:sequence>
1568     <xs:attribute name="id" type="xs:ID" />
1569     <xs:attribute name="itemID" type="IDType" />
1570 </xs:complexType>
1571
1572
  
```

1573 5.2. <ModifyResponse> element

1574 The <ModifyResponse> element contains the <Status> element, which describes whether or not the requested
 1575 modification succeeded. There is also a possible time stamp attribute, which provides a time value that can be used
 1576 later to check whether there have been any changes since this modification, and an itemIDRef attribute to map the
 1577 <ModifyResponse> elements to the <Modify> elements in the request.

1578 The schema for <ModifyResponse>

```

1579     <xs:element name="ModifyResponse" type="ResponseType" />
1580 <xs:complexType name="ResponseType">
1581     <xs:sequence>
1582         <xs:element ref="Status" />
1583         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1584     </xs:sequence>
1585     <xs:attribute name="id" type="xs:ID" />
1586     <xs:attribute name="itemIDRef" type="IDReferenceType" />
1587     <xs:attribute name="timeStamp" type="xs:dateTime" />
1588 </xs:complexType>
1589
1590
  
```

1591 5.3. Processing Rules

1592 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

1593 The <Modify> can contain multiple <Modification> elements. The following rules specify how those must be
 1594 supported and handled:

- 1595 • A WSP MUST support one <Modification> element inside a <Modify> and SHOULD support multiple. If
 1596 the <Modify> contains multiple <Modification> elements and the WSP supports only one <Modification>
 1597 element inside a <Modify>, the processing of the whole <Modify> MUST fail and a status code indicating failure
 1598 MUST be returned in the response. The value NoMultipleAllowed SHOULD be used for the second level
 1599 status code. If a WSP supports multiple <Modification> element inside a <Modify>, it MAY register the
 1600 urn:liberty:dst:multipleModification discovery option keyword.
- 1601 • If the processing of a <Modification> fails even partly due to some reason, depending on the service and/or a
 1602 WSP either the processing of the whole <Modify> MUST fail or a WSP MUST try to achieve partial success.
 1603 The top level status code Failed or Partial MUST be used to indicate the failure (complete or partial) and a
 1604 more detailed second level status code SHOULD be used to indicate the reason for failing to completely process
 1605 the failed <Modify> element. Furthermore, the ref attribute of the <Status> element SHOULD carry the value
 1606 of the itemID of the failed <Modification> element and in partial success cases it MUST carry this value. The
 1607 modifications made based on already processed <Modification> elements of the <Modify> MUST be rolled
 1608 back in case of a complete failure. A WSP MUST NOT support multiple <Modification> elements inside one
 1609 <Modify>, if it cannot roll back and partial failure is not allowed.

- 1610 • When multiple `<Modification>` elements inside one `<Modify>` element are supported and partial success is
1611 allowed, a WSC MUST use the `itemID` attribute in each `<Modification>` element so that a WSP can identify
1612 the failed parts, when it is returning status information for a partial success.
- 1613 What is modified and how depends on a number of parameters including the value of the `<Select>` element, the
1614 content of the provided `<NewData>` element, the value of the `overrideAllowed` attribute, and the current content of
1615 the underlying conceptual XML document.
- 1616 The following rules specify in more detail how modification works:
- 1617 • When adding new data, the `<Select>` element will point in the conceptual XML document to an element which
1618 does not exist yet. The new element is added as a result of processing the `<Modification>` element. In such
1619 cases, when the ancestor elements of the new element do not exist either, they MUST be added as part of processing
1620 of the `<Modification>` element so that processing could be successful.
- 1621 • If the `<Select>` points to multiple places and there is a `<NewData>` element with new values, the processing of
1622 the `<Modification>` MUST fail because it is not clear where to store the new data. If there is no `<NewData>`
1623 element and the `overrideAllowed` attribute is set to *True*, then the processing of `<Modification>` can continue
1624 normally, because it is acceptable to delete multiple data elements at once (for example, all `AddressCards`).
1625 When the `overrideAllowed` is set to *False* or is missing, the `<NewData>` element MUST be present as new
1626 data should be added. If the `<NewData>` element is missing in this case, the processing of the `<Modification>`
1627 MUST fail and the second level status code `MissingNewDataElement` SHOULD be returned in addition to top
1628 level status code.
- 1629 • When there is the `<NewData>` element with new values and the `<Select>` points to existing information, the
1630 processing of the `<Modification>` MUST fail, if the `overrideAllowed` attribute is not set to *True*. When
1631 the `overrideAllowed` attribute does not exist or is set to *False*, the new data in the `<NewData>` element can
1632 only be accepted in two cases: either there is no existing element to which the `<Select>` points, or there can be
1633 multiple data elements of the same type. This means that, if the `<Select>` points to an existing container element,
1634 which has a subelement, and only one such container element can exist, the `<Modification>` MUST fail, even
1635 if the only subelement the container element has inside the `<NewData>` does not yet exist in the conceptual XML
1636 document. The second level status code `ExistsAlready` SHOULD be used to indicate in details the reason for
1637 the failure in addition to the top level status code. The lack of those other sub-elements inside the `<NewData>`
1638 means that they should be removed, which is only possible when `overrideAllowed` attribute equals to *True*.
1639 When there can be multiple elements of the same type, the addition of a new element MUST fail, if there exists
1640 already an element of same type have the same value of the distinguishing part. In the case of a personal profile
1641 service, adding a new `<AddressCard>` element MUST fail, if there already exists an `<AddressCard>` element
1642 which has an `id` attribute of the same value as the provided new `<AddressCard>` element. The second level status
1643 code `ExistsAlready` SHOULD also be used to indicate the detailed reason for failure.
- 1644 • When all or some of the data inside the `<NewData>` element is not supported by the WSP, or the provided data is not
1645 valid, the processing of the whole `<Modification>` SHOULD fail and second level status code `InvalidData`
1646 SHOULD be returned in the response.
- 1647 • When the `<Modification>` element tries to extend the service either by pointing to a new data type behind an
1648 `<Extension>` element with the `<Select>` element, or having new sub-elements under an `<Extension>` element
1649 inside the `<NewData>` element and the WSP does not support extension in general or for the requesting party, it
1650 SHOULD be indicated in the response message with the second level status code `ExtensionNotSupported`.
1651 When the WSP supports extensions, but does not accept the content of the `<Select>` or `<NewData>`, then second
1652 level status codes `InvalidSelect` and `InvalidData` SHOULD be used as already described.
- 1653 The common attributes belonging to the attribute groups `commonAttributes` and `leafAttributes` are mainly
1654 supposed to be written by the WSP hosting the data service. There are some additional rules for handling these
1655 common attributes in case of modifications.

- 1656 • When any of the `ACC`, `modifier`, `ACCTime` or `modificationTime` attributes is used in a resource, the WSP
1657 hosting the data service MUST keep their values up to date. When data is modified, the `modifier` MUST contain
1658 the `ProviderID` of the modifier or have no value, and the `modificationTime` MUST define the time of the
1659 modification or have no value. The `ACC` MUST define the attribute collection context of the current value of a data
1660 element or have no value and the `ACCTime` MUST define the time, when the current value of the `ACC` was defined
1661 or have no value.
- 1662 • If the `<NewData>` contains `modifier`, `modificationTime` or `ACCTime` attributes for any data element, the WSP
1663 MUST ignore these and update the values based on other information than those attributes inside the `<NewData>`
1664 provided by the WSC. If the `ACC` attribute is included for any data element, the WSP MAY accept it, depending
1665 on how much it trusts the requesting service provider. The WSP MAY also accept the `id` attribute provided inside
1666 the `<NewData>` and some services MAY require that the `id` attribute MUST be provided by the requesting service
1667 provider.
- 1668 • The `id` attribute MUST NOT be used as a global unique identifier. The value MUST be chosen so that it works
1669 only as unique identifier inside the conceptual XML document, and the value of the `id` attribute SHOULD be kept
1670 the same even if the element is otherwise modified. A WSP MAY not even allow changing the value of the `id`
1671 attribute or any other attribute used to distinguish elements with the same name from each other.
- 1672 • When data is modified based on the `<Modify>` request, the values of the `modificationTime` attributes written
1673 by the WSP hosting the data service MAY be same for all inserted and updated elements, but there is no guarantee
1674 that they will be exactly the same. When the `modificationTime` attribute is used by a data service, the WSP
1675 MUST keep it up to date to indicate the time of the latest modification of an element and update it, when ever
1676 a modification is done either using the `<Modify>` request or some other way. When the `modificationTime`
1677 attribute is used in container elements, the time of a modification MUST be propagated to all ancestor elements of
1678 the modified element all the way up to the root element.
- 1679 Accounting for concurrent updates is handled using the `notChangedSince` attribute inside the `<Modification>`
1680 element.
- 1681 • When the `notChangedSince` attribute is present, the modifications specified by the `<Modification>` element
1682 MUST NOT be made, if any part of the data to be modified has changed since the time specified by the
1683 `notChangedSince` attribute.
1684 The second level status code `ModifiedSince` MUST be used to indicate that the modification was not done
1685 because the data has been modified since the time specified by the `notChangedSince` attribute. If a WSP does
1686 not support processing of this attribute properly, it MUST NOT make any changes and it MUST return the second
1687 level status code `ChangeHistoryNotSupported`. If a WSP supports this `notChangedSince` attribute, it MUST
1688 also support the `changedSince` attribute of the `<QueryItem>` element.
- 1689 A WSC might not be allowed to make certain modifications or any modifications at all.
- 1690 • When a WSP processes the `<Modification>`, it MUST check, whether the resource owner (for example, the
1691 Principal) has given consent to the requester to modify the data. To be able to check WSC-specific access
1692 rights, the WSP MUST authenticate the WSC (see [\[LibertySecMech\]](#) and [\[LibertyMetadata\]](#)). If the consent
1693 check fails for any part of the requested data, the WSP MUST NOT make the modifications requested in the
1694 `<Modification>` element, even when such consent is missing only for some subelement or attribute. The WSP
1695 MAY try to get consent from the Principal while processing the request perhaps using an interaction service (for
1696 more information see [\[LibertyInteract\]](#)). The processing of the `<Modification>` element MUST fail, if the
1697 modification was not allowed. The second level status code `ActionNotAuthorized` MAY be used, if it is
1698 considered that the privacy of the owner of the resource is not compromised. A WSP might check the access rights
1699 at a higher level, before getting to DST processing and MAY return an ID-* Fault Message [\[LibertySOAPBinding\]](#)
1700 and not process the `<Modify>` element at all, if the requesting WSC is not allowed to modify the data.

1701 The WSP may have some restrictions for the data it is hosting.

- 1702 • The schemas for different data services may have some elements for which there is not an exact upper limit on how
 1703 many can exist. For practical reasons, implementations may set some limits. If a request tries to add more elements
 1704 than a WSP supports, the WSP will not accept the new element(s) and the processing of the <Modification>
 1705 element MUST fail. The WSP should use a second level status code NoMoreElements to indicate this specific
 1706 case.
- 1707 • The schemas for different data services may not specify the length of elements and attributes especially in the
 1708 case of strings. The WSP may also have limitations of this kind. If a request tries to add longer data elements
 1709 or attributes than a WSP supports, the WSP may not accept the data and the processing of the <Modification>
 1710 element will fail. The WSP should use a second level status code DataTooLong to indicate this specific case.

1711 5.4. Examples

1712 Please note that as the modification examples are based on the [\[LibertyIDPP\]](#) all DST elements are also in the
 1713 namespace defined in [\[LibertyIDPP\]](#).

1714 This example adds a home address to the personal profile of a Principal:

```

1715 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1716   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1717   <pp:Modification>
1718     <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
1719     <pp:NewData>
1720       <pp:AddressCard id='98123'>
1721         <pp:AddressType>urn:liberty:pp:addrType:home</pp:AddressType>
1722         <pp:Address>
1723           <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way North</pp:PostalAddress>
1724           <pp:PostalCode>98503-2341</pp:PostalCode>
1725           <pp:L>Olympia</pp:L>
1726           <pp:ST>wa</pp:ST>
1727           <pp:C>us</pp:C>
1728         </pp:Address>
1729       </pp:AddressCard>
1730     </pp:NewData>
1731   </pp:Modification>
1732 </pp:Modify>
1733
1734
1735
1736
```

1737 The following example replaces the current home address with a new home address in the personal profile of a
 1738 Principal. Please note that this request will fail if there are two or more home addresses in the profile, because it
 1739 is not clear in this request, which of those addressed should be replaced by this address. In such a case the id attribute
 1740 should be used to explicitly point which of the addresses should be changed.

```

1741 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1742   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1743   <pp:Modification overrideAllowed="True">
1744     <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:
1745     home']</pp:Select>
1746     <pp:NewData>
1747       <pp:AddressCard id="98123">
1748         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1749         <pp:Address>
1750           <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way</pp:PostalAddress>
1751         </pp:Address>
1752       </pp:AddressCard>
1753     </pp:NewData>
1754   </pp:Modification>
1755 </pp:Modify>

```

```

1753         <pp:PostalCode>98503-2342</pp:PostalCode>
1754         <pp:L>Olympia</pp:L>
1755         <pp:ST>wa</pp:ST>
1756         <pp:C>us</pp:C>
1757     </pp:Address>
1758 </pp:AddressCard>
1759 </pp:NewData>
1760 </pp:Modification>
1761 </pp:Modify>
1762
1763
  
```

1764 This example replaces the current address identified by an id of '98123' with a new home address, if that address has
 1765 not been modified since 12:40:01 21th January 2003 UTC.

```

1766
1767
1768 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1769   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1770   <pp:Modification notChangedSince="2003-01-21T12:40:01Z" overrideAllowed="True">
1771     <pp>Select>/pp:PP/pp:AddressCard[@pp:id='98123']</pp>Select>
1772     <pp:NewData>
1773       <pp:AddressCard id="98123">
1774         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1775         <pp:Address>
1776           <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way South</pp:PostalAddress>
1777           <pp:PostalCode>98503-2398</pp:PostalCode>
1778           <pp:L>Olympia</pp:L>
1779           <pp:ST>wa</pp:ST>
1780           <pp:C>us</pp:C>
1781         </pp:Address>
1782       </pp:AddressCard>
1783     </pp:NewData>
1784   </pp:Modification>
1785 </pp:Modify>
1786
1787
  
```

1788 The following example adds another home address to the personal profile of a Principal. An id is provided for the
 1789 new address.

```

1790
1791
1792 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1793   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1794   <pp:Modification>
1795     <pp>Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:home']</pp>Select>
1796     <pp:NewData>
1797       <pp:AddressCard id="12398">
1798         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1799         <pp:Address>
1800           <pp:PostalAddress>1234 Beach Way</pp:PostalAddress>
1801           <pp:PostalCode>98765-1234</pp:PostalCode>
1802           <pp:L>Olympia</pp:L>
1803           <pp:ST>wa</pp:ST>
1804           <pp:C>us</pp:C>
1805         </pp:Address>
1806       </pp:AddressCard>
1807     </pp:NewData>
1808   </pp:Modification>
1809 </pp:Modify>
1810
1811
1812
  
```

1813 The following example removes all current home addresses from the personal profile of a Principal:

```
1814
1815
1816 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1817   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1818   <pp:Modification overrideAllowed="True">
1819     <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:
1820 home']</pp:Select>
1821   </pp:Modification>
1822 </pp:Modify>
1823
1824
```

1825 The response for a valid <Modify> is as follows:

```
1826
1827
1828 <pp:ModifyResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1829 timeStamp="2003-03-23T03:40:00Z">
1830   <pp:StatusCode="OK"/>
1831 </pp:ModifyResponse>
1832
1833
```

1834 6. Subscriptions and Notifications

1835 The subscriptions are a mechanism through which WSCs can request for notifications when specified event happens.
1836 The basic case is subscribing to change notifications to get updates when the data hosted by a data service related to a
1837 Principal changes. A WSC may subscribe to change notifications even before the data exists. E.g. a WSC may want to
1838 know, when a Principal adds an email address to her profile, and get that email address to be able to send email to the
1839 Principal. The change of data is not the only possible reason for a notification, there can be service specific triggers
1840 for notifications, e.g. periodic notifications containing current values and notifications after a Principal switches on
1841 her terminal.

1842 As the notifications do not only reveal the data they are carrying, but also that certain thing has just happened, WSPs
1843 must be even more careful to make sure they honor the privacy of the Principals.

1844 Subscriptions and notifications have two basic elements: `<Subscription>` and `<Notification>`. These basic
1845 elements used by subscription and notification protocols are introduced first. After that the protocols for subscriptions
1846 and notifications are defined.

1847 6.1. Basic elements

1848 The basic elements `<Subscription>` and `<Notification>` are transferred using different protocol messages and
1849 can have slightly different content depending on a message, e.g. all the parameters of the `<Subscription>` element
1850 are not needed in each message. The basic elements are defined here, but their use is defined together with protocol
1851 elements.

1852 6.1.1. `<Subscription>` element

1853 The `<Subscription>` element contains all the parameters for a subscription. It defines, what data a WSC wants to
1854 have, where it should be sent, when a subscription expires, which events should trigger notifications, etc.

1855 The first parameter inside the `<Subscription>` element is the `<Select>` element. This is the same element used
1856 also when querying and modifying data. It defines, what data a notification should return. The use of the `<Select>`
1857 element inside the `<Subscription>` element might be a bit different than its use when querying and modifying. The
1858 specifications for services MUST specify possible differences.

1859 The same `<ChangeFormat>` element, which is used for querying for changes, can be used, when subscribing to
1860 change notifications. This element is used to indicate, which change formats a WSC supports and which it prefers.

1861 The element `<NotifyTo>` contains the information a WSP needs to be able to send the requested notifications to
1862 a requesting WSC. The type definition for the `<NotifyTo>` element is imported from [\[LibertySOAPBinding\]](#) and
1863 the element contains three different types of subelements. The `<sb-ext:Endpoint>` element contains the address
1864 to which notifications MUST be sent. There can also be one or more `<sb-ext:SecurityMechID>` elements
1865 indicating which security mechanisms a WSP may use (it MUST NOT use any other). A WSC may also provide
1866 credentials, which a WSP MUST use, when sending notifications. The credentials are provided using one or more
1867 `<sb-ext:Credential>` elements. Note that the type definition imported from [\[LibertySOAPBinding\]](#) also contains
1868 attributes. Attributes `S.actor` and `S.mustunderstand` MUST NOT be used in a `<NotifyTo>` element. A WSC
1869 can provide a WSP different information for sending normal notifications and for sending notifications telling that the
1870 subscription has ended. The information for sending end notifications is provided in a `<NotifyEndedTo>` element,
1871 which is optional and has same type definition as the `<NotifyTo>` element. Please note that attributes `S.actor`
1872 and `S.mustunderstand` MUST NOT be used in a `<NotifyEndedTo>` element either. If the `<NotifyEndedTo>`
1873 element is not used, the same information provided for normal notifications is also used for end notifications. The
1874 purpose of the `<NotifyEndedTo>` element is to make it possible to receive notifications in one point and manage
1875 subscriptions in another point, when end notifications are used.

1876 There can be different type of notifications. E.g. a notification can be sent immediately or multiple notification could
1877 be sent in a bigger batch. The element `<Type>` defines, what type of notifications a WSC is requesting. The element

1878 <Type> element is of type element `TypeType` and this type MUST be specified by services including the detailed
 1879 semantics and allowed values.

1880 The normal reason for a notification is that the data addressed by the <Select> element has changed. There can be
 1881 also other reasons triggering notifications. The <Trigger> element contains those triggers. The <Trigger> element
 1882 is of type element `TriggerType`. This type MUST be defined in the services schemas and the service specifications
 1883 MUST define semantics and values for this parameter. When the <Trigger> element is not used, a WSC is requesting
 1884 normal change notifications unless otherwise specified by a service specification.

1885 A subscription is not valid forever. The `starts` attribute defines the time after which a subscription is valid and
 1886 notifications can be sent, if the triggering event occurs. The `starts` attribute MUST be used only, when a subscription
 1887 is not supposed to be valid immediately after processing the request. The `expires` attribute defines the time, when a
 1888 subscription expires, if not renewed before that time. Instead of the `expires` attribute the `duration` attribute may
 1889 be used to indicate the duration of a subscription.

1890 The different subscriptions related to a same resource are distinguished from each other by IDs. A WSC uses
 1891 the `invokeID` attribute, when creating a new subscription. After creating a subscription, it is referred by the
 1892 `subscriptionID` attribute. A WSP gives value to this attribute.

1893 Usually a notification contains data related to a resource. Sometimes a notification could be used to indicate that
 1894 an event related to a resource has happened, e.g. the data addressed by the <Select> element has changed, without
 1895 containing the changed data. The attribute `includeData` defines, should the data be included or not. The default value
 1896 is `Yes`. Other possible values are: `No` (no data is returned) and `YesWithCommonAttributes` (the data is returned
 1897 with the common attributes).

1898 The use of the different parameters of a subscription are defined in more detail with the protocol elements and
 1899 processing rules related to those. The schema for the <Subscription> element is as follows:

```

1900 <!-- Subscription Element -->
1901 <xs:element name="Subscription">
1902   <xs:complexType>
1903     <xs:sequence>
1904       <xs:annotation>
1905         <xs:documentation>
1906           NOTE: The SelectType, TypeType, and TriggerType below
1907           must be defined by the schema that includes this one.
1908         </xs:documentation>
1909       </xs:annotation>
1910       <xs:element name="Select" type="SelectType" minOccurs="0"/>
1911       <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
1912       <xs:element name="NotifyTo" type="sbext:ServiceInstanceUpdateType" minOccurs="0"/>
1913       <xs:element name="NotifyEndedTo" type="sbext:ServiceInstanceUpdateType" minOccurs="0"/>
1914       <xs:element name="Type" type="TypeType" minOccurs="0"/>
1915       <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
1916       <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1917     </xs:sequence>
1918     <xs:attribute name="starts" type="xs:dateTime"/>
1919     <xs:attribute name="expires" type="xs:dateTime"/>
1920     <xs:attribute name="duration" type="xs:duration"/>
1921     <xs:attribute name="id" type="xs:ID"/>
1922     <xs:attribute name="invokeID" type="IDType"/>
1923     <xs:attribute name="subscriptionID" type="IDType"/>
1924     <xs:attribute name="includeData" default="Yes">
1925       <xs:simpleType>
1926         <xs:restriction base="xs:string">
1927           <xs:enumeration value="Yes"/>
1928           <xs:enumeration value="No"/>
1929           <xs:enumeration value="YesWithCommonAttributes"/>
1930         </xs:restriction>
1931       </xs:simpleType>
1932     </xs:attribute>
1933   </xs:complexType>
  
```

1934 </xs:element>
 1935
 1936

1937 **6.1.2. <Notification> element**

1938 The <Notification> element is the basic element, when either a normal notification or notification to indicate the
 1939 end of a subscription is sent. The main content of the <Notification> element is the <Data> element, which
 1940 contains the data the notification carries, e.g. the current location or the changed home address. In a case of a change
 1941 notification same formats as in responses to queries for changed data are used.

1942 The <Data> element may also contain some other type of data indicating what kind of an event has happened. The
 1943 whole <Data> element might not be used at all as it is possible to subscribe to notifications to indicate that an event
 1944 has happened, e.g. data has changed without having the data in the notification message.

1945 In addition to the <Data> element the <Notification> element has a number of attributes. There are two ID
 1946 attributes: `invokeID` and `subscriptionID`. The `invokeID` attribute is only used to in the initial reply to the
 1947 subscription and after that the `subscriptionID` attribute is used.

1948 The `changeFormat` attribute **MUST** be used to indicate the used format for data changes in case of a change
 1949 notification unless only one format is specified to be used for the service type in question.

1950 The `expires` attribute is used to indicate in a notification message the time, when the subscription will expire.
 1951 Instead of the `expires` attribute the `duration` attribute may be used to indicate, how long the subscription is
 1952 still valid. In an end notification the `endReason` attribute can be used to indicate the reason for the end of the
 1953 subscription. This might give some indication to a WSC that a WSP is having some problems or whether it makes
 1954 sense or not for a WSC to try to make the subscription again. The `endReason` attribute is not used in normal
 1955 notifications, only when notifying that a subscription has ended. Possible values for the `endReason` attribute include:
 1956 `urn:liberty:dst:endreason:unspecified`, `urn:liberty:dst:endreason:wscnotacknowledging`,
 1957 `urn:liberty:dst:endreason:resourcedeleted`, `urn:liberty:dst:endreason:expired` and
 1958 `urn:liberty:dst:endreason:credentialexpired`. A WSP must be careful not to compromise the
 1959 privacy of a Principal, when sending the reason codes for ending a subscription.

1960 The schema for the <Notification> element is as follows:

```

1961 <!-- Notification Element -->
1962 <xs:element name="Notification">
1963   <xs:complexType>
1964     <xs:sequence>
1965       <xs:element name="Data" minOccurs="0">
1966         <xs:complexType>
1967           <xs:sequence>
1968             <xs:any minOccurs="0" maxOccurs="unbounded"/>
1969           </xs:sequence>
1970         </xs:complexType>
1971       </xs:element>
1972     </xs:sequence>
1973     <xs:attribute name="id" type="xs:ID"/>
1974     <xs:attribute name="invokeID" type="IDType"/>
1975     <xs:attribute name="subscriptionID" type="IDType" use="required"/>
1976     <xs:attribute ref="changeFormat"/>
1977     <xs:attribute name="expires" type="xs:dateTime"/>
1978     <xs:attribute name="duration" type="xs:duration"/>
1979     <xs:attribute name="endReason" type="xs:anyURI"/>
1980   </xs:complexType>
1981 </xs:element>
1982
```

1983 **6.2. Protocol elements**

1984 There are eight different protocol elements defined in following subchapters, half of them are responses to
1985 the others. A WSC may send the <Subscribe> and <QuerySubscription> elements and receive back the
1986 <SubscribeResponse> and <Subscriptions> elements. Based on existing subscriptions a WSP may sent
1987 to a WSC <Notify> and <Ended> elements and may get back as acknowledgements <NotifyResponse> and
1988 <EndedResponse> elements.

1989 **6.2.1. Subscribing notifications and modifying subscriptions**

1990 **6.2.1.1. <Subscribe> element**

1991 The <Subscribe> element is used to subscribe to notifications, modify existing subscriptions, renew subscriptions
1992 which are about to expire and cancel subscriptions which a WSC does not need any more.

1993 The <Subscribe> element has two main parts, first it has either the <ResourceID> element or the
1994 <EncryptedResourceID> element to specify the resource in question and then the <Subscription> ele-
1995 ments. One <Subscribe> element can have multiple <Subscription> elements related to the same resource.
1996 The <Subscribe> element has also the id and itemID attributes to be used in the same way as they are used in
1997 other request messages. Also one new attribute is defined: returnCurrentValues, the default value is True. If a
1998 WSC doesn't want to get the current values of the data as part of the response to the <Subscribe>, it can set it to
1999 False. Please note that there are cases in which this attribute is meaningless. If the start time has been defined for a
2000 subscription, no current values are returned by the response.

2001 The content of the <Subscription> elements depends, what a WSC wants to do. The rules are simple:

- 2002 • For new subscription, out of invokeID and subscriptionID attributes there is only the invokeID attribute.
2003 The subscriptionID attribute MUST NOT be used, when a WSC requests for a new subscription. The rest of
2004 parameters are used normally.
- 2005 • For renewing an existing subscription the subscriptionID attribute MUST be used to identify the right
2006 subscription and the expires or the duration attribute MUST be used to indicate the requested lifetime of
2007 the subscription. The id MAY be used, if needed. As renewing can be combined with other modifications to an
2008 existing subscription, other content is also allowed except the invokeID attribute, which MUST NOT be used.
- 2009 • For canceling an existing subscription, a WSC MUST use the subscriptionID attribute to identify the
2010 subscription and no other content is allowed, except the id MAY be used, if needed.
- 2011 • For modifying an existing subscription, the subscriptionID attribute MUST be used to identify the right
2012 subscription and the changed information MUST be provided. If any of the elements, <Select>, <Type>,
2013 <Trigger> or <Extension> has changed, the element must be provided with the requested current content.
2014 This means that if one of the triggers has changed and two other triggers haven't, they all must be provided as
2015 part of the <Trigger> element. In case some of those four elements doesn't have any content anymore, an empty
2016 element must be provided to indicate the deletion, e.g. <Trigger/>. If there is no changes e.g. to <Trigger>,
2017 the <Trigger> element is not present in the message.

2018 The schema for the <Subscribe> element is as follows:

```

2019 <!-- Subscribing notifications and modifying, renewing and deleting existing notifications -->
2020 <xs:element name="Subscribe" type="SubscribeType"/>
2021 <xs:complexType name="SubscribeType">
2022   <xs:sequence>
2023     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2024     <xs:element ref="Subscription" maxOccurs="unbounded"/>
2025     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2026   </xs:sequence>
2027   <xs:attribute name="id" type="xs:ID"/>
2028   <xs:attribute name="itemID" type="IDType"/>
2029   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
2030 </xs:complexType>
2031
2032
```

2033 6.2.1.2. <SubscribeResponse> element

2034 The response to the <Subscribe> has two main parts. There are the <Status> element and the <Notification>
 2035 element(s). The <Notification> element is allowed to have the invokeID attribute only, when it is the response
 2036 to the initial subscription, after that there is no invokeID associated with the subscription as the subscriptionID
 2037 is used. So, when a WSP is responding to subscription modifications, renewal and deletion, there MUST NOT be an
 2038 invokeID attribute. The timeStamp attribute is provided to be used e.g. for the value of the changedSince attribute
 2039 in <Query> after the subscription has expired.

2040 The schema for the <SubscribeResponse> element is as follows:

```

2041 <xs:element name="SubscribeResponse" type="SubscribeResponseType"/>
2042 <xs:complexType name="SubscribeResponseType">
2043   <xs:sequence>
2044     <xs:element ref="Status"/>
2045     <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
2046     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2047   </xs:sequence>
2048   <xs:attribute name="id" type="xs:ID"/>
2049   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2050   <xs:attribute name="timeStamp" type="xs:dateTime"/>
2051 </xs:complexType>
2052
2053
```

2054 6.2.1.3. Processing rules

2055 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

2056 The same <Subscribe> element can be used for a number of tasks: to create a new subscription, to renew, modify
 2057 and cancel an existing subscription.

- 2058 • When a WSC has included the invokeID attribute in a <Subscription> element, this <Subscription>
 2059 element is used to create a new subscription. A WSC MUST NOT put a subscriptionID attribute into the
 2060 same <Subscription> element. If both the subscriptionID and the invokeID attributes are in the same
 2061 <Subscription> element sent by a WSC, the receiving WSP MUST discard that <Subscription> element
 2062 and SHOULD use in the response the second level status code NewOrExisting to indicate that it has failed to
 2063 process the <Subscription> element as it has not been clear does a WSC want to create a new subscription or
 2064 modify an existing subscription.

- 2065 • When a new subscription is created, a WSP MUST return the `invokeID` attribute with the same value as the
2066 requesting WSC used for it. A WSP MUST also create a new ID for the subscription to be used in the notifications
2067 and further requests. This ID MUST be returned in the `subscriptionID` attribute of the `<Notification>`
2068 element return in the `<SubscribeResponse>` related to the `<Subscribe>` element in the request message.
- 2069 • When a WSC wants to cancel an existing subscription, the `<Subscription>` element MUST have only the
2070 `subscriptionID` attribute to indicate, which subscriptions a WSC wants to cancel. An `ID` attribute MAY
2071 also be used, if needed. If a WSC wants to cancel all subscriptions related to the same resource, it MUST
2072 use an empty `<Subscription>` element (i.e. `<Subscription/>` element) which MUST NOT have the
2073 `subscriptionID` attribute. Again an `ID` attribute MAY also be used, if needed. If any other content in addition
2074 to the `subscriptionID` attribute and possible `id` attribute is used, a WSP MUST process the `<Subscription>`
2075 element as it would be a request to modify or renew an existing subscriptions.
- 2076 • When a WSC wants to modify an existing subscription, it MUST send a `<Subscription>` element which has
2077 the `subscriptionID` referring to an existing subscription. Only the changed values MUST be included in the
2078 `<Subscription>` element. E.g. if a WSC does not want to change the type of the subscription, it does not
2079 need to include the `<Type>` element. The existing values of a subscription for the parameters included in the
2080 `<Subscription>` element MUST be overwritten with the values provided in the `<Subscription>` element, if
2081 a WSP supports the provided new values (see rest of the processing rules in this chapter and also the common
2082 processing rules in [Section 3](#)).
- 2083 For each successful `<Subscription>` element in the request a WSP MUST return a `<Notification>` element in
2084 the response, even when no data is returned to confirm the expiration time/duration of the subscription.
- 2085 • A `<Notification>` element MUST have the `subscriptionID` of the subscription it is related to. If the
2086 `<Subscription>` element was sent to cancel all existing subscription, the corresponding `<Notification>`
2087 element in the `<SubscribeResponse>` MUST NOT have the `subscriptionID` attribute at all.
- 2088 • The attribute `endReason` MUST NOT be used in a `<Notification>` element, when sending it in a
2089 `<SubscribeResponse>` element. If a subscription was not accepted, the reason is indicated with status codes.
- 2090 A subscription must contain the address to which notifications should be sent and other information needed by a WSP
2091 to be able to send the requested notifications.
- 2092 • When a new subscription is created, the processing of the subscription MUST fail if there is no `<NotifyTo>`
2093 element in the `<Subscription>` element. The second level status code `MissingNotifyToElement` SHOULD
2094 be used in addition to the top level status code.
- 2095 • The `<NotifyTo>` element MUST contain at least the `<Endpoint>` element, which MUST contain a complete
2096 URI. If the `<Endpoint>` element is missing from a `<NotifyTo>` element, the processing of the `<Subscription>`
2097 containing this `<NotifyTo>` element MUST fail and a second level status code `MissingEndpointElement`
2098 indicating this SHOULD be used in addition to the top level status code. Also, if a WSP is able to verify the validity
2099 of the address provided in the `<Endpoint>` element and finds out that the address is not valid, the processing of
2100 the `<Subscription>` MUST fail and a second level status code `InvalidEndpoint` indicating this SHOULD be
2101 used in addition to the top level status code.

- 2102 • A WSC SHOULD also provide one or more <SecurityMechID> elements to indicate, which security mech-
2103 anisms it supports. The <SecurityMechID> elements MUST be specified according to [LibertySecMech]
2104 and the mechanisms SHOULD be listed in the order of preference by the WSC. If a WSC does not provide
2105 <SecurityMechID> element, the default is urn:liberty:security:2003-08:null:null, which MUST be
2106 avoided, when releasing privacy sensitive data (the case almost every time with identity services). A WSP MUST
2107 refuse to use mechanisms, which it considers to be not good enough, also Principal owning the resource may have
2108 set some security requirements, which a WSP MUST follow. If a WSP wants to get the <SecurityMechID>
2109 element, but it is not included, the processing of the <Subscription> element MUST fail and a second level
2110 status code MissingSecurityMechIDElement indicating this SHOULD be used in addition to the top level
2111 status code. If a WSP does not support or accept any of the proposed security mechanisms, it MUST NOT accept
2112 the subscription and so the processing of the <Subscription> element MUST fail and a second level status
2113 code SecurityMechIDNotAccepted indicating this SHOULD be used in addition to the top level status code.
2114 Some of the security mechanisms require also credentials and a WSC MUST provide those also. If credentials are
2115 needed, but they are not provided, the processing of the <Subscription> element MUST fail and a second level
2116 status code MissingCredentials indicating this SHOULD be used in addition to the top level status code.
- 2117 • The information provided in a <NotifyTo> element is also used, when sending end notifications unless separate
2118 <NotifyEndedTo> element is provided. The information in a provided <NotifyEndedTo> element replaces
2119 the information in a <NotifyTo> element. Please note that a <NotifyEndedTo> element might contain only a
2120 different endpoint for end notifications, in which case the same security mechanisms and credentials provided in
2121 a <NotifyTo> element are used. The processing rules for the subelements of a <NotifyEndedTo> element are
2122 the same as for subelements of a <NotifyTo> element except the <Endpoint> element is not mandatory as the
2123 value provided in a <NotifyTo> element may be used (see previous paragraph).
- 2124 • Please note that if needed credentials expire earlier than a subscription is suppose to expire and a WSC does
2125 not provide new credentials before they expire, the subscription will expire as a WSP is not able to send the
2126 notifications anymore.
- 2127 There might be different type of notifications and different triggers causing those. A WSP may not support all different
2128 type of features available.
- 2129 • A WSP MUST follow the processing rules defined in the specification for the service a WSP hosting for the
2130 elements <Type> and <Trigger>. If the use of these elements is not specified for the service, but either
2131 of both of them are included in a <Subscription> element in a <Subscribe> request, the processing of
2132 the <Subscription> MUST fail and a second level status code indicating this SHOULD be used, either
2133 TypeNotSupported or TriggerNotSupported.
- 2134 • If a WSP does not support the <Type> of the notification a WSC requests, the processing of the <Subscription>
2135 MUST fail and a second level status code TypeNotSupported indicating this SHOULD be used in addition to the
2136 top level status code. Similarly if a WSP does not support the <Trigger> a WSC requests, the processing of the
2137 <Subscription> MUST fail and a second level status code TriggerNotSupported indicating this SHOULD
2138 be used in addition to the top level status code.
- 2139 A WSC may request, when the first notification may be sent and when a subscription should expire.
- 2140 • If a <Subscription> element contains a starts attribute, that subscription, if accepted, MUST be valid after
2141 the time defined by this starts attribute. If there is no starts attribute used, then that subscription, if accepted
2142 by a WSP, MUST be valid immediately after processing the request. Also, if the time specified by the starts
2143 attribute is in the past, then that subscription, if accepted by a WSP, MUST be valid immediately after processing
2144 the request.

-
- 2145 • The time specified by the `expires` attribute MUST be the same time or a later time than the time specified by the
2146 `starts` attribute in the same `<Subscription>` element. It also MUST be later than the current time. If either of
2147 the checks is not passed, then the processing of the `<Subscription>` MUST fail and a second level status code
2148 `InvalidExpires` indicating this SHOULD be used in addition to the top level status code.
- 2149 • A WSP MAY change the time when a subscription expires from the expiration time requested by a WSC with
2150 the `expires` or `duration` attribute. A WSP MAY shorten the expiration time, but it MUST NOT make the
2151 expiration time longer. If no `expires` or `duration` attribute is not included in a `<Subscription>` element in a
2152 `<Subscribe>` request from a WSC, a WSP MUST specify the expiration time for the subscription, if expiration
2153 times are required either by the service specification or the WSP. A WSP MUST use the same attribute a WSC
2154 used in a request to specify the expiration time (`expires` or `duration`), unless otherwise specified in a service
2155 specification. A service specification MAY define that only either of those may be used.
- 2156 • If a WSC wants to renew an existing subscription, which is about to end, it MUST modify that subscription and
2157 give a new value for the `expires` or `duration` attribute of that subscription. A WSP MAY modify the new value
2158 in the same way as it MAY modify the proposed value for a new subscription.
- 2159 • When expiration times are used, a WSP MUST include the `expires` or `duration` attribute with the current value
2160 in each `<Notification>` element included in a `<SubscribeResponse>` element.
- 2161 • There are two special cases available using subscriptions expirations. When the `starts` and the `expires`
2162 attributes have exactly same values, the meaning is that a notification MUST be sent exactly at that time. When the
2163 `duration` attribute is set to zero (e.g. "POD"), the subscription expires immediately after first notification. Note:
2164 when a WSC wants to create subscription, which is valid only for a very short time it MUST explicitly specify a
2165 very short duration and not try to use a value equally to zero.
- 2166 A WSC may want to get the current values a the data when subscribing to notifications.
- 2167 • The default value of the `returnCurrentValues` attribute is `Yes`, so by default a WSP MUST include the current
2168 values in the response. If a WSC sets the value of the `returnCurrentValues` to `No`, a WSP MUST NOT return
2169 the current values in the response. Also if the `includeData` in the `<Subscription>` element is set to `No`, the
2170 data for that subscription MUST NOT be included in the response even, if the `returnCurrentValues` attribute
2171 has the value `Yes`.
- 2172 • The current values of the data addressed by the `<Select>` element of a `<Subscription>` element are included in
2173 the `<Data>` element of the corresponding `<Notification>` element. The data inside the `<Data>` element MUST
2174 be selected in the same way as in a normal query (see [Section 4](#)). If the `includeData` in the `<Subscription>`
2175 has the value `YesWithCommonAttributes`, the existing common attributes MUST be returned with the data.
- 2176 • A WSP MUST NOT return the `<Data>` element, when a WSC is modifying, renewing or canceling existing
2177 subscription unless a WSC is modifying the `<Select>` element of a subscription.
- 2178 The access and privacy policies specified by the resource owner may not allow a WSC to subscribe to the data of a
2179 resource or to some events related to a resource.

- 2180 • When a WSP processes a <Subscription> element, it MUST check whether the resource owner (the Principal,
2181 for example) has given consent to return the requested information in notification messages. To be able to check
2182 WSC-specific access rights, the WSP MUST authenticate the WSC (see [LibertySecMech] and [LibertyMeta-
2183 data]). The WSP MUST also check that any usage directive given in the request is acceptable based on the usage
2184 directives defined by the resource owner (see [LibertySOAPBinding]). If either check fails, the WSP MUST NOT
2185 accept the subscription and the processing of that <Subscription> MUST fail. The WSP MAY try to get consent
2186 from the Principal while processing the request, perhaps by using an interaction service (see [LibertyInteract]). A
2187 WSP might check the access rights and policies in usage directives at a higher level, before getting to DST pro-
2188 cessing and MAY, in this case, just return an ID-* Fault Message [LibertySOAPBinding] without processing the
2189 <Subscribe> element at all, if the requesting WSC is not allowed to subscribe to data or event in question.
- 2190 • Note that there can be consent for subscribing to some data element, but not its attributes. A Principal might not
2191 want to release the modifier attribute, if she does not want to reveal information about which services she uses.
2192 If a WSC is not allowed to get all the data, but some data it wants, a WSP SHOULD accept the subscription, but it
2193 MAY also reject it. If a subscription is accepted, the data for which there is no consent from the Principal MUST
2194 be handled as if there was no data, i.e. that data MUST NOT be returned in the response message, even if the
2195 current values should be returned. Also that data MUST NOT be included in the notification messages sent later
2196 on.
- 2197 • If a WSC has made a subscription and included the usage directive it has promised to obey and later wants to
2198 change the usage directive, it MUST cancel the subscription and make a new subscription with the new value for
2199 the usage directive.

2200 6.2.2. Querying existing subscriptions

2201 6.2.2.1. <QuerySubscriptions> element

2202 The existing subscriptions can also be queried. The purpose is to list all the currently active subscriptions the requesting
2203 WSC has related to the specific resource. Either the <ResourceID> element or the <EncryptedResourceID>
2204 element is used to identify the resource.

2205 The schema for the <QuerySubscriptions> element is as follows:

```
2206 <!-- Query subscriptions -->  
2207 <xs:element name="QuerySubscriptions" type="QuerySubscriptionsType"/>  
2208 <xs:complexType name="QuerySubscriptionsType">  
2209   <xs:sequence>  
2210     <xs:group ref="ResourceIDGroup" minOccurs="0"/>  
2211     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>  
2212   </xs:sequence>  
2213   <xs:attribute name="id" type="xs:ID"/>  
2214   <xs:attribute name="itemID" type="IDType"/>  
2215 </xs:complexType>  
2216  
2217
```

2218 6.2.2.2. <Subscriptions> element

2219 The response to the <QuerySubscriptions> is a list of <Subscription> elements inside a <Subscriptions>
2220 element. The invokeID MUST NOT be used in <Subscription> elements returned.

2221 The <Status> element is also included in the <Subscriptions> element to indicate, how the processing of the
2222 <QuerySubscriptions> succeeded.

2223 The schema for the <Subscriptions> element is as follows:

```
2224 <xs:element name="Subscriptions" type="SubscriptionsType"/>  
2225 <xs:complexType name="SubscriptionsType">
```

```
2226     <xs:sequence>
2227         <xs:element ref="Status"/>
2228         <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded"/>
2229         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2230     </xs:sequence>
2231     <xs:attribute name="id" type="xs:ID"/>
2232     <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2233 </xs:complexType>
2234
2235
```

2236 6.2.2.3. Processing rules

2237 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

- 2238 • A WSP MUST return in a response to a <QuerySubscriptions> all valid subscriptions a WSC has related to
2239 the specified resource at the WSP. One <Subscription> element MUST be used for each valid subscription the
2240 request WSC has.
- 2241 • If a WSC does not have any valid subscriptions at a WSP related to the specified resource, the WSP MUST NOT
2242 return any <Subscription> element in the <Subscriptions> element.
- 2243 • A <Subscription> element in a <Subscriptions> element MUST have all the parameters currently valid for
2244 it.
- 2245 • A <Subscription> element in a <Subscriptions> element MUST have a subscriptionID attribute and
2246 MUST NOT have a invokeID attribute as a invokeID is not valid after the response to the original request to
2247 create a new subscription.

2248 6.2.3. Sending notifications

2249 6.2.3.1. <Notify> element

2250 The subscriptions are made to get the notifications about data changes and other events. The <Notify> element
2251 is used to carry the <Notification> elements. A <Notification> element inside a <Notify> element is not
2252 allowed to have the endReason attribute as the subscription has not ended. Ending must be indicated with the Ended
2253 message. The timeStamp attribute is provided to be used e.g. for the value of the notChangedSince attribute in
2254 <Modify> after the subscription has expired.

2255 The schema for the <Notify> element is as follows:

```
2256 <!-- Sending Notifications and Notifying about subscriptions which have ended-->
2257
2258 <xs:element name="Notify" type="NotifyEndedType"/>
2259
2260 <xs:complexType name="NotifyEndedType">
2261     <xs:sequence>
2262         <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
2263         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2264     </xs:sequence>
2265     <xs:attribute name="id" type="xs:ID"/>
2266     <xs:attribute name="itemID" type="IDType"/>
2267     <xs:attribute name="timeStamp" type="xs:dateTime"/>
2268 </xs:complexType>
2269
2270
```

2271 6.2.3.2. <NotifyResponse> element

2272 The response to a <Notify> acknowledges the received <Notification> elements and so it just contains the
2273 <Status> element.

2274 The <Notify> messages may not always be acknowledged. A service specification MUST define are these
2275 acknowledgements used or not or is it an implementation/deployment specific decision.

2276 The schema for the <NotifyResponse> element is as follows:

```
2277 <xs:element name="NotifyResponse" type="NotifyEndedResponseType"/>
2278
2279 <xs:complexType name="NotifyEndedResponseType">
2280 <xs:sequence>
2281 <xs:element ref="Status"/>
2282 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2283 </xs:sequence>
2284 <xs:attribute name="id" type="xs:ID"/>
2285 <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2286 </xs:complexType>
2287
2288
```

2289 6.2.3.3. Processing rules

2290 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

2291 • A WSP MUST send a notification message to a WSC, which has made a subscription, when an event defined by
2292 the parameters of that subscription happens.

2293 • When sending a notification message to a WSC, a WSP MUST use the information provided in the <NotifyTo>
2294 element (endpoint, security mechanism and credentials).

2295 • If the receiving WSC can not process successfully one of the <Notification> elements inside one <Notify>
2296 element, it SHOULD process normally the rest of the <Notification> elements and try to achieve a partial
2297 success. A WSC MUST support multiple <Notification> elements inside one <Notify> element.

2298 • A <Notification> element inside a notification message MUST NOT have an invokeID or a endReason
2299 attribute. If there are a WSC MUST discard them.

2300 • A <Notification> element inside a notification message MUST have a subscriptionID attribute to iden-
2301 tify the subscription based on which the notification message is sent. If the subscriptionID attribute
2302 is missing, the processing of that <Notification> element MUST fail and a second level status code
2303 MissingSubscriptionID SHOULD be used in addition to a top level status code. If a WSC does not rec-
2304 ognize the value of a subscriptionID attribute, the processing of that <Notification> element MUST fail
2305 and a second level status code InvalidSubscriptionID SHOULD be used in addition to a top level status code.

2306 • A <Notification> element inside a notification message MUST have the expires or the duration attribute,
2307 when subscription expiration is used. When a WSC receiving a notification knows that the expires or
2308 the duration attribute should have been used, but it is not, it SHOULD use the second level status code
2309 MissingExpiration. A WSC MUST decide is this a failure or not, but it SHOULD anyway indicate to a
2310 WSP that it was expecting the expires or the duration attribute.

- 2311 • If a <Notification> element is supposed to contain data about a resource (i.e. the includeData attribute of a
2312 subscription has either the value Yes or YesWithCommonAttributes), the <Data> element MUST be used in a
2313 <Notification> element. The content of a <Data> element MUST be according to the parameters of the related
2314 subscription and the related event, which has caused this <Notification> element to be sent inside a notification
2315 message. In case of a change notification the same formatting rules for the content as in case of a query for changes
2316 MUST be followed (see [Section 4](#)). A WSP MUST NOT include any data, which the WSC is not allowed to get
2317 based on access rights and privacy policies defined by the resource owner. If a <Data> element should have
2318 been included in a <Notification> element, but it is missing, the processing of the <Notification> element
2319 MUST fail and a second level status code MissingDataElement SHOULD be used in addition to the top level
2320 status code.
- 2321 • For change notification a changeFormat attribute MUST be added for a <Data> element to indicate the format
2322 used to shown the changes, if a service specification has not mandated only one specific format to be used for this.
- 2323 • If the data inside a <Data> element is invalid, the processing of the <Notification> element MUST fail and a
2324 second level status code InvalidData SHOULD be used in addition to the top level status code. A WSC MUST
2325 accept all the data, which can be consider as possible normal extension, if extensions are allowed for a service
2326 based on the service specification.
- 2327 • A WSP SHOULD resend a notification for which it does not get an acknowledgement in reasonable time, if
2328 acknowledgements are used. If a WSP does not get acknowledgments at all within its time and other limits, it
2329 MAY cancel the related subscription.

2330 6.2.4. Notifying the end of a subscription

2331 6.2.4.1. <Ended> element

2332 There is also another kind of a notification message. The <Ended> message indicates that the subscription has ended
2333 for one reason or another. The content of the <Ended> is the same as for the <Notify> with the difference that now
2334 the endReason attribute must be present in the <Notification> element(s) and no <Data> element is used.

2335 The schema for the <Ended> element is as follows:

```
2336 <xs:element name="Ended" type="NotifyEndedType"/>  
2337  
2338
```

2339 6.2.4.2. <EndedResponse> element

2340 The response to the <Ended> is similar to the response to the <Notify>, so mainly only the <Status> element.

2341 The schema for the <EndedResponse> element is as follows:

```
2342 <xs:element name="EndedResponse" type="NotifyEndedResponseType"/>  
2343  
2344
```

2345 6.2.4.3. Processing rules

2346 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

- 2347 • When a subscription is not anymore valid, a WSP MUST send an end notification to indicate this, if end
2348 notifications are used. The information provided in the <NotifyEndedTo> element of the subscription MUST be
2349 used. If the <NotifyEndedTo> element has not been used or it doesn't contain all the information, the information
2350 provided in the <NotifyTo> element of the subscription is used to complement.

-
- 2351 • A <Notification> element inside an end notification message MUST have the attributes endReason and
2352 subscriptionID and it MAY also have the id attribute, if needed, but it MUST NOT have any other content. If
2353 it has, the receiving WSC SHOULD ignore the other content.
- 2354 • The value of an endReason attribute SHOULD be either urn:liberty:dst:endreason:unspecified,
2355 urn:liberty:dst:endreason:expired, urn:liberty:dst:endreason:credentialsexpired,
2356 urn:liberty:dst:endreason:wscnotacknowledging or urn:liberty:dst:endreason:resourcedeleted.
2357 It MAY also have some service or implementation specific value. A WSP MUST be careful not to use any values,
2358 which might compromise the privacy of a Principal.
- 2359 • A <Notification> element inside an end notification message MUST have a subscriptionID attribute
2360 to identify the subscription which has ended. If the subscriptionID attribute is missing, the process-
2361 ing of that <Notification> element MUST fail and a second level status code MissingSubscriptionID
2362 SHOULD be used in addition to a top level status code. If a WSC does not recognize the value of a
2363 subscriptionID attribute, the processing of that <Notification> element MUST fail and a second level
2364 status code InvalidSubscriptionID SHOULD be used in addition to a top level status code.
- 2365 • A WSP SHOULD resend an end notification for which it does not get an acknowledgement in reasonable time, if
2366 acknowledgements are used.

2367 7. The Schema for the DST Protocols

```
2368
2369     <?xml version="1.0" encoding="UTF-8"?>
2370 <xs:schema
2371     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2372     xmlns:disco="urn:liberty:disco:2003-08"
2373     elementFormDefault="qualified"
2374     attributeFormDefault="unqualified"
2375     xmlns:sbext="urn:liberty:sb:2004-04">
2376
2377     <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>
2378     <xs:import namespace="urn:liberty:disco:2003-08" schemaLocation="liberty-idwsf-disco-sv
2379 c-v1.2.xsd"/>
2380     <xs:import namespace="urn:liberty:sb:2
2381 004-04" schemaLocation="liberty-idwsf-soap-binding-exts-v1.2.xsd"/>
2382
2383     <xs:annotation>
2384     <xs:documentation>
2385 The source code in this XSD file was excerpted verbatim from:
2386
2387 Liberty ID-WSF Data Services Template Specification
2388 Version 2.0
2389 23 March 2005
2390
2391 NOTE: This schema must be used within the context of another schema -
2392 It is not intended to validate by itself.
2393
2394 The scheme which includes this must provide definitions for:
2395 TypeType
2396 SelectType
2397 TriggerType
2398
2399     Copyright (c) 2004-2005 Liberty Alliance participants, see
2400     http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php
2401
2402     </xs:documentation>
2403 </xs:annotation>
2404
2405 <xs:element name="ResourceID" type="disco:ResourceIDType"/>
2406 <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>
2407 <xs:group name="ResourceIDGroup">
2408     <xs:choice>
2409         <xs:element ref="ResourceID"/>
2410         <xs:element ref="EncryptedResourceID"/>
2411     </xs:choice>
2412 </xs:group>
2413
2414 <xs:element name="ChangeFormat">
2415     <xs:simpleType>
2416         <xs:restriction base="xs:string">
2417             <xs:enumeration value="ChangedElements"/>
2418             <xs:enumeration value="CurrentElements"/>
2419         </xs:restriction>
2420     </xs:simpleType>
2421 </xs:element>
2422
2423 <xs:attribute name="changeFormat">
2424     <xs:simpleType>
2425         <xs:restriction base="xs:string">
2426             <xs:enumeration value="ChangedElements"/>
2427             <xs:enumeration value="CurrentElements"/>
2428             <xs:enumeration value="All"/>
2429         </xs:restriction>
2430     </xs:simpleType>
2431 </xs:attribute>
2432
```

```

2433 <!-- Querying Data -->
2434 <xs:element name="Query" type="QueryType" />
2435 <xs:complexType name="QueryType">
2436   <xs:sequence>
2437     <xs:group ref="ResourceIDGroup" minOccurs="0" />
2438     <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded">
2439       <xs:complexType>
2440         <xs:sequence>
2441           <xs:annotation>
2442             <xs:documentation>
2443               NOTE: The below two types (SelectType and SortType) must
2444               be defined by the schema that includes this one.
2445             </xs:documentation>
2446           </xs:annotation>
2447           <xs:element name="Select" type="SelectType" minOccurs="0" />
2448           <xs:element name="Sort" type="SortType" minOccurs="0" />
2449           <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2" />
2450         </xs:sequence>
2451         <xs:attribute name="id" type="xs:ID" />
2452         <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0" />
2453         <xs:attribute name="itemID" type="IDType" />
2454         <xs:attribute name="changedSince" type="xs:dateTime" />
2455         <xs:attribute name="count" type="xs:nonNegativeInteger" />
2456         <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0" />
2457         <xs:attribute name="setID" type="IDType" />
2458         <xs:attribute name="setReq">
2459           <xs:simpleType>
2460             <xs:restriction base="xs:string">
2461               <xs:enumeration value="Static" />
2462               <xs:enumeration value="DeleteSet" />
2463             </xs:restriction>
2464           </xs:simpleType>
2465         </xs:attribute>
2466       </xs:complexType>
2467     </xs:element>
2468     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2469   </xs:sequence>
2470   <xs:attribute name="id" type="xs:ID" />
2471   <xs:attribute name="itemID" type="IDType" />
2472 </xs:complexType>
2473
2474 <xs:element name="QueryResponse" type="QueryResponseType" />
2475 <xs:complexType name="QueryResponseType">
2476   <xs:sequence>
2477     <xs:element ref="Status" />
2478     <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
2479       <xs:complexType>
2480         <xs:sequence>
2481           <xs:any minOccurs="0" maxOccurs="unbounded" />
2482         </xs:sequence>
2483         <xs:attribute name="id" type="xs:ID" />
2484         <xs:attribute name="itemIDRef" type="IDReferenceType" />
2485         <xs:attribute name="notSorted">
2486           <xs:simpleType>
2487             <xs:restriction base="xs:string">
2488               <xs:enumeration value="Now" />
2489               <xs:enumeration value="Never" />
2490             </xs:restriction>
2491           </xs:simpleType>
2492         </xs:attribute>
2493         <xs:attribute ref="changeFormat" />
2494         <xs:attribute name="remaining" type="xs:integer" />
2495         <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0" />
2496         <xs:attribute name="setID" type="IDType" />
2497       </xs:complexType>
2498     </xs:element>
2499     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />

```

```

2500     </xs:sequence>
2501     <xs:attribute name="id" type="xs:ID"/>
2502     <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2503     <xs:attribute name="timeStamp" type="xs:dateTime"/>
2504 </xs:complexType>
2505
2506 <!-- Modifying Data -->
2507 <xs:element name="Modify" type="ModifyType"/>
2508 <xs:complexType name="ModifyType">
2509   <xs:sequence>
2510     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2511     <xs:element name="Modification" maxOccurs="unbounded">
2512       <xs:complexType>
2513         <xs:sequence>
2514           <xs:annotation>
2515             <xs:documentation>
2516               NOTE: The below SelectType must be defined by
2517               the schema that includes this one.
2518             </xs:documentation>
2519           </xs:annotation>
2520           <xs:element name="Select" type="SelectType" minOccurs="0"/>
2521           <xs:element name="NewData" minOccurs="0">
2522             <xs:complexType>
2523               <xs:sequence>
2524                 <xs:any minOccurs="0" maxOccurs="unbounded"/>
2525               </xs:sequence>
2526             </xs:complexType>
2527           </xs:element>
2528         </xs:sequence>
2529         <xs:attribute name="id" type="xs:ID"/>
2530         <xs:attribute name="itemID" type="IDType"/>
2531         <xs:attribute name="notChangedSince" type="xs:dateTime"/>
2532         <xs:attribute name="overrideAllowed" type="xs:boolean" default="0"/>
2533       </xs:complexType>
2534     </xs:element>
2535     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2536   </xs:sequence>
2537   <xs:attribute name="id" type="xs:ID"/>
2538   <xs:attribute name="itemID" type="IDType"/>
2539 </xs:complexType>
2540
2541 <xs:element name="ModifyResponse" type="ResponseType"/>
2542 <xs:complexType name="ResponseType">
2543   <xs:sequence>
2544     <xs:element ref="Status"/>
2545     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2546   </xs:sequence>
2547   <xs:attribute name="id" type="xs:ID"/>
2548   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2549   <xs:attribute name="timeStamp" type="xs:dateTime"/>
2550 </xs:complexType>
2551
2552 <!-- Subscribing notifications and modifying, renewing and deleting existing notifications -->
2553 <xs:element name="Subscribe" type="SubscribeType"/>
2554 <xs:complexType name="SubscribeType">
2555   <xs:sequence>
2556     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2557     <xs:element ref="Subscription" maxOccurs="unbounded"/>
2558     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2559   </xs:sequence>
2560   <xs:attribute name="id" type="xs:ID"/>
2561   <xs:attribute name="itemID" type="IDType"/>
2562   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
2563 </xs:complexType>
2564
2565 <xs:element name="SubscribeResponse" type="SubscribeResponseType"/>
2566 <xs:complexType name="SubscribeResponseType">

```

```

2567     <xs:sequence>
2568         <xs:element ref="Status"/>
2569         <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
2570         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2571     </xs:sequence>
2572     <xs:attribute name="id" type="xs:ID"/>
2573     <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2574     <xs:attribute name="timeStamp" type="xs:dateTime"/>
2575 </xs:complexType>
2576
2577 <!-- Query subscriptions -->
2578 <xs:element name="QuerySubscriptions" type="QuerySubscriptionsType"/>
2579 <xs:complexType name="QuerySubscriptionsType">
2580     <xs:sequence>
2581         <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2582         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2583     </xs:sequence>
2584     <xs:attribute name="id" type="xs:ID"/>
2585     <xs:attribute name="itemID" type="IDType"/>
2586 </xs:complexType>
2587
2588 <xs:element name="Subscriptions" type="SubscriptionsType"/>
2589 <xs:complexType name="SubscriptionsType">
2590     <xs:sequence>
2591         <xs:element ref="Status"/>
2592         <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded"/>
2593         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2594     </xs:sequence>
2595     <xs:attribute name="id" type="xs:ID"/>
2596     <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2597 </xs:complexType>
2598
2599 <!-- Subscription Element -->
2600 <xs:element name="Subscription">
2601     <xs:complexType>
2602         <xs:sequence>
2603             <xs:annotation>
2604                 <xs:documentation>
2605                     NOTE: The SelectType, TypeType, and TriggerType below
2606                     must be defined by the schema that includes this one.
2607                 </xs:documentation>
2608             </xs:annotation>
2609             <xs:element name="Select" type="SelectType" minOccurs="0"/>
2610             <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
2611             <xs:element name="NotifyTo" type="sbext:ServiceInstanceUpdateType" minOccurs="0"/>
2612             <xs:element name="NotifyEndedTo" type="sbext:ServiceInstanceUpdateTyp
2613 e" minOccurs="0"/>
2614             <xs:element name="Type" type="TypeType" minOccurs="0"/>
2615             <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
2616             <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2617         </xs:sequence>
2618         <xs:attribute name="starts" type="xs:dateTime"/>
2619         <xs:attribute name="expires" type="xs:dateTime"/>
2620         <xs:attribute name="duration" type="xs:duration"/>
2621         <xs:attribute name="id" type="xs:ID"/>
2622         <xs:attribute name="invokeID" type="IDType"/>
2623         <xs:attribute name="subscriptionID" type="IDType"/>
2624         <xs:attribute name="includeData" default="Yes">
2625             <xs:simpleType>
2626                 <xs:restriction base="xs:string">
2627                     <xs:enumeration value="Yes"/>
2628                     <xs:enumeration value="No"/>
2629                     <xs:enumeration value="YesWithCommonAttributes"/>
2630                 </xs:restriction>
2631             </xs:simpleType>
2632         </xs:attribute>
2633     </xs:complexType>

```

```

2634     </xs:element>
2635
2636     <!-- Sending Notifications and Notifying about subscriptions which have ended-->
2637
2638     <xs:element name="Notify" type="NotifyEndedType" />
2639
2640     <xs:complexType name="NotifyEndedType" >
2641         <xs:sequence>
2642             <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />
2643             <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2644         </xs:sequence>
2645         <xs:attribute name="id" type="xs:ID" />
2646         <xs:attribute name="itemID" type="IDType" />
2647         <xs:attribute name="timeStamp" type="xs:dateTime" />
2648     </xs:complexType>
2649
2650     <xs:element name="NotifyResponse" type="NotifyEndedResponseType" />
2651
2652     <xs:complexType name="NotifyEndedResponseType" >
2653         <xs:sequence>
2654             <xs:element ref="Status" />
2655             <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2656         </xs:sequence>
2657         <xs:attribute name="id" type="xs:ID" />
2658         <xs:attribute name="itemIDRef" type="IDReferenceType" />
2659     </xs:complexType>
2660
2661     <xs:element name="Ended" type="NotifyEndedType" />
2662
2663     <xs:element name="EndedResponse" type="NotifyEndedResponseType" />
2664
2665     <!-- Notification Element -->
2666     <xs:element name="Notification" >
2667         <xs:complexType>
2668             <xs:sequence>
2669                 <xs:element name="Data" minOccurs="0" >
2670                     <xs:complexType>
2671                         <xs:sequence>
2672                             <xs:any minOccurs="0" maxOccurs="unbounded" />
2673                         </xs:sequence>
2674                     </xs:complexType>
2675                 </xs:element>
2676             </xs:sequence>
2677             <xs:attribute name="id" type="xs:ID" />
2678             <xs:attribute name="invokeID" type="IDType" />
2679             <xs:attribute name="subscriptionID" type="IDType" use="required" />
2680             <xs:attribute ref="changeFormat" />
2681             <xs:attribute name="expires" type="xs:dateTime" />
2682             <xs:attribute name="duration" type="xs:duration" />
2683             <xs:attribute name="endReason" type="xs:anyURI" />
2684         </xs:complexType>
2685     </xs:element>
2686 </xs:schema>
2687
2688

```

2689 **8. Checklist for Service Specifications**

2690 The following table provides a checklist of issues which should be addressed by individual service type specifications.
2691 Such specifications should always state which optional features of the DST they support, in addition to defining more
2692 general things such as discovery option keywords and the `SelectType` XML type used by the service type. A service
2693 specification should complete this table with the specific values and statements required by the specification.

2694 For optional features, the language specified by [\[RFC2119\]](#) **MUST** be used to define whether these features are
2695 available for implementations and deployments. For example, specifying that a feature 'MAY' be implemented by
2696 a WSP means that WSPs may or may not support the feature, and that WSCs should be ready to handle both cases.

2697

Table 3. General Service Parameters (1/2)

| Parameter | Value |
|-----------------------------------|---|
| <ServiceType> | The <ServiceType> URN (see [LibertyDisco]). For example: urn:liberty:id-sis-pp:2003-08 |
| Discovery Options | The discovery option keywords (see [LibertyDisco]) can either be listed with semantics here, or via a reference to the correct chapter in the specification. Please note that the DST defines the following discovery option keywords and the service specification must list which of these the service may use: <pre style="text-align: center;">urn:liberty:dst:allPaths urn:liberty:dst:can:extend urn:liberty:dst:changeHistorySupported urn:liberty:dst:extend urn:liberty:dst:fullXPath urn:liberty:dst:multipleResources urn:liberty:dst:multipleQueryItems urn:liberty:dst:multipleModification urn:liberty:dst:noModify urn:liberty:dst:noPagination urn:liberty:dst:noQuery urn:liberty:dst:noQuerySubscriptions urn:liberty:dst:noSorting urn:liberty:dst:noStatic urn:liberty:dst:noSubscribe</pre> |
| Data Schema | A reference to the services full XML schema should be provided here. |
| SelectType Definition | The full type definition of the <Select> element, or a reference to the definition in the specification. For example: <pre style="text-align: center;"><xs:simpleType name="SelectType"> <xs:restriction base="xs:string"/> </xs:simpleType></pre> |
| Semantics of the <Select> element | The semantics of the SelectType should be given or referenced here. Some examples include: MUST support Restricted XPath (see chapter X.Y for the set required), MAY extend the required set to cover all paths, MAY support full XPATH. |

2698

Table 4. General Service Parameters (2/2)

| Parameter | Value |
|--------------------------|---|
| Element uniqueness | State here how elements with the same name are distinguished from each other. For example, the <code>id</code> attribute MUST be used for <code><AddressCard></code> and <code><MsgContact></code> elements, <code>xml:lang</code> and <code>script</code> attributes used for localized elements. |
| Data Extension Supported | State here whether extension is supported and if so, describe this support. A reference to the specification chapter defining this can be given. E.g. New elements and discovery option keywords MAY be defined, see chapter Y.X for more details. |

2699

Table 5. Query Parameters (1/2)

| Parameter | Value |
|--|--|
| Support querying | Some services or implementations may or may not support querying data. This should be stated here. E.g. an implementation SHOULD support querying data. |
| Multiple <code><Query></code> elements | Are multiple <code><Query></code> elements supported? |
| Multiple <code><QueryItem></code> elements | Are multiple <code><QueryItem></code> elements supported? |
| Support sorting | Is sorting supported? If it is, the sorting criteria must be specified here or a reference added to the definition of the available criterias. |
| SortType definition | The full type definition of the <code><Sort></code> element, or a reference to the definition. E.g. when sorting is not supported the type definition using Liberty utility schema could be: <pre> <xs:complexType name="SortType"> <xs:complexContent> <xs:restriction base="EmptyType" /> </xs:complexContent> </xs:complexType> </pre> |
| Support <code>changedSince</code> | State here whether the <code>changedSince</code> attribute is supported. (for example, this service SHOULD support <code>changedSince</code>) |
| Supported formats | If the <code>changedSince</code> attribute is supported, there a three different formats available: <code>ChangedElements</code> , <code>CurrentElements</code> and <code>All</code> . The supported formats MUST be listed here and possible default value MUST also be stated here, if such is chosen. |

2700

Table 6. Query Parameters (2/2)

| Parameter | Value |
|--|---|
| Support <code>includeCommonAttributes</code> | State whether the <code>includeCommonAttributes</code> attribute is supported. (MUST be, or SHOULD be for example) |
| Support pagination | Some services or implementations may or may not support pagination. This should be stated here. If the pagination is supported, it MUST be listed for which elements the pagination is supported. E.g. An implementation SHOULD support pagination for <code><x></code> and <code><y></code> elements and MAY support it for all other elements. |
| Support static sets | When pagination is supported, some services or implementations may or may not support static sets to handle concurrent access. This should be stated here. If static sets are supported, it MUST be listed for which elements they are supported. E.g. An implementation SHOULD support static sets for element <code><x></code> and MAY support it for all other elements. |
| <code><Extension></code> in <code><Query></code> | Is the <code><Extension></code> element inside the <code><Query></code> element used? If so, for what purpose? |

2701

Table 7. Modify Parameters

| Parameter | Value |
|---|---|
| Support modification | Some services or implementations may or may not support modifications. This should be stated here. |
| Multiple <code><Modify></code> elements | If modifications are supported, are multiple <code><Modify></code> elements supported? |
| Multiple <code><Modification></code> elements | If modifications are supported, are multiple <code><Modification></code> elements supported? |
| Support partial success | If multiple <code><Modification></code> elements are supported, is partial success supported or are only atomic modifications allowed? |
| Support <code>notChangedSince</code> | State here whether the <code>notChangedSince</code> attribute is supported. (for example, this service SHOULD support <code>notChangedSince</code>) |
| <code><Extension></code> in <code><Modify></code> | Is the <code><Extension></code> element inside the <code><Modify></code> element used? If so, for what purpose? For the purpose a reference to some other chapter can be given. |

2702

Table 8. Subscribe Parameters (1/2)

| Parameter | Value |
|--|---|
| Support subscribing to notifications | Some services or implementations may or may not support subscribing to notifications. This should be stated here. E.g. an implementation SHOULD support subscribing to notifications. |
| Use of the <Subscribe> element for modifying and renewing subscriptions. | The <Subscribe> element may be used for subscribing notifications, renewing subscriptions, canceling subscriptions and modifying existing subscriptions. A service specification may state that modifying and renewing are not supported, if so, it must be stated here. E.g. Modifying existing subscriptions MUST NOT be supported, but renewing MUST be supported. |
| Multiple <Subscribe> elements | If subscriptions are supported, are multiple <Subscribe> elements supported? |
| Multiple <Subscription> elements | If subscriptions are supported, are multiple <Subscription> elements supported? |
| Use of the <NotifyEndedTo> element | If end notifications are used (see later table), a WSC may e.g. request those to be sent to a different end point than normal notifications and also request a WSP to use different credentials, when sending end notifications, if <NotifyEndedTo> element is supported. The support of the <NotifyEndedTo> element MUST be stated here. E.g. Not applicable as end notifications are not used. |
| TypeType definition | <p>The full type definition of the <Type> element, or a reference to the definition. E.g. when the <Type> element is not used the type definition using Liberty utility schema could be:</p> <pre data-bbox="925 1312 1339 1459"> <xs:complexType name="TypeType"> <xs:complexContent> <xs:restriction base="EmptyType" /> </xs:complexContent> </xs:complexType> </pre> <p>A reference to the right place in the service specification discussing the semantics and processing rules related to the <Type> element MUST be added here, if the element is used.</p> |

2703

Table 9. Subscribe Parameters (2/2)

| Parameter | Value |
|--|---|
| TriggerType definition | <p>The full type definition of the <Trigger> element, or a reference to the definition. E.g. when the <Trigger> element is not used the type definition using Liberty utility schema could be:</p> <pre style="text-align: center;"> <xs:complexType name="TriggerType"> <xs:complexContent> <xs:restriction base="EmptyType" /> </xs:complexContent> </xs:complexType> </pre> <p>A reference to the right place in the service specification discussing the semantics and processing rules related to the <Trigger> element MUST be added here, if the element is used.</p> |
| Start of a subscription | <p>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 <code>starts</code> attribute. It MUST be specified here, is the <code>starts</code> attribute supported or not. E.g. The <code>starts</code> attribute MUST NOT be used.</p> |
| Subscription expiration | <p>Usually subscriptions expire after a certain time, but a service specification may also specify e.g. that subscription expiration is not used and WSCs must cancel subscriptions after they are not needed. It MUST be specified here, do subscriptions expire or not. E.g. Subscription expiration MUST be used.</p> |
| Use of expires and duration attributes | <p>Two different attributes, <code>expires</code> and <code>duration</code>, are defined to be used by a WSC to specify the requested lifetime of a subscription and by a WSP to communication the actual lifetime of a subscription. A service specification MUST state are both or only either of them used, if subscription expiration is used. E.g. Both <code>expires</code> and <code>duration</code> MAY be used.</p> |
| Support expires==starts | <p>Is it allowed to specify same time both for the <code>starts</code> and the <code>expires</code> attribute to request one notification message at a specified time. E.g. same value MAY be used both for the <code>starts</code> and the <code>expires</code> attribute.</p> |
| Support zero duration | <p>Is it allowed to specify the <code>duration</code> to have a zero value to have a subscription valid only for one notification. E.g. zero value for <code>duration</code> MUST NOT be used for the <code>duration</code> attribute.</p> |
| <Extension> in <Subscribe> | <p>Is the <Extension> element inside the <Subscribe> element used? If so, for what purpose? For the purpose a reference to some other chapter can be given.</p> |

2704

Table 10. QuerySubscriptions Parameters

| Parameter | Value |
|---|---|
| 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. |
| Multiple <QuerySubscriptions> elements | If subscriptions are supported, are multiple <QuerySubscriptions> elements supported? |
| <Extension> in <QuerySubscriptions> | Is the <Extension> element inside the <QuerySubscriptions> element used? If so, for what purpose? For the purpose a reference to some other chapter can be given. |

2705

Table 11. Notify Parameters

| Parameter | Value |
|--------------------------------|---|
| Support notifications | Some services or implementations may or may not support notifications. This should be stated here. E.g. an implementation SHOULD support notifications. |
| Are notifications acknowledged | Some services or implementations may or may not support acknowledging notifications using <NotifyResponse>. This should be stated here. E.g. Notifications MUST BE acknowledge. |
| <Extension> in <Notify> | Is the <Extension> element inside the <Notify> element used? If so, for what purpose? For the purpose a reference to some other chapter can be given. |

2706

Table 12. EndNotify Parameters

| Parameter | Value |
|------------------------------------|--|
| Support end notifications | Some services or implementations may or may not support end notifications. This should be stated here. E.g. MUST NOT be supported. |
| Are end notifications acknowledged | Some services or implementations may or may not support acknowledging end notifications using <EndedResponse>. This should be stated here. E.g. End notifications MUST BE acknowledge. |
| <Extension> in <Ended> | Is the <Extension> element inside the <Ended> element used? If so, for what purpose? For the purpose a reference to some other chapter can be given. |

2707 **References**

2708 **Normative**

- 2709 [LibertyDisco] Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification," Version 1.2, Liberty
2710 Alliance Project (12 December 2004). <http://www.projectliberty.org/specs>
- 2711 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.1, Liberty
2712 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 2713 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, eds. "Liberty ID-WSF SOAP Binding Specification
2714 ," Version 1.2, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 2715 [LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 1.1, Liberty
2716 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 2717 [LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 1.1, Liberty Alliance
2718 Project (14 December 2004). <http://www.projectliberty.org/specs>
- 2719 [LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.2, Liberty Alliance Project
2720 (14 December 2004). <http://www.projectliberty.org/specs>
- 2721 [LibertyGlossary] "Liberty Technical Glossary," Version 1.4, Liberty Alliance Project (14 Dec 2004).
2722 <http://www.projectliberty.org/specs> Hodges, Jeff, eds.
- 2723 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.0, Liberty Alliance Project (12
2724 November 2003). <http://www.projectliberty.org/specs>
- 2725 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May
2726 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium
2727 <http://www.w3.org/TR/xmlschema-1/>
- 2728 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
2729 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt> [March 1997].
- 2730 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible
2731 Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium
2732 <http://www.w3.org/TR/2000/REC-xml-20001006>

2733 **Informative**

- 2734 [LibertyIDPP] Kellomaki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.0, Liberty
2735 Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 2736 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
2737 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>