



Liberty ID-WSF Data Services Template Specification

Version: v2.0-06

Editors:

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

Contributors:

Robert Aarts, Nokia Corporation
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, Entrust
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: draft-liberty-idwsf-dst-v2.0-06.pdf

Copyright © 2004 Liberty Alliance Project

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 ActivCard; America Online, Inc.; American Express Travel Related Services; Axalto; Bank of
16 America Corporation; Bell Canada; Cingular Wireless; Cisco Systems, Inc.; Communicator, Inc.; Deloitte & Touche
17 LLP; Earthlink, Inc.; Electronic Data Systems, Inc.; Entrust, Inc.; Epok, Inc.; Ericsson; Fidelity Investments; France
18 Telecom; Gemplus; General Motors; Hewlett-Packard Company; i2 Technologies, Inc.; Internet2; Intuit Inc.;
19 MasterCard International; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nextel Communications; Nippon
20 Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OneName Corporation;
21 Openwave Systems Inc.; Phaos Technology; Ping Identity Corporation; PricewaterhouseCoopers LLP; RegistryPro,
22 Inc.; RSA Security Inc; Sabre Holdings Corporation; SAP AG; SchlumbergerSema; Sigaba; SK Telecom; Sony
23 Corporation; Sun Microsystems, Inc.; Symlabs, Inc.; Trustgenix; United Airlines; VeriSign, Inc.; Visa International;
24 Vodafone Group Plc; Wave Systems. 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 **Revision History**

32 **Revision: 01 Date:** 06 April 2004

33 Added the subscriptions and notifications and also sorting and pagination. Modified the structure a bit to avoid
34 repeating same processing rules again and again for each request message.

35 **Revision: 02 Date:** 07 July 2004

36 Updated based on feedback in Barcelona. Some clarifications in error/status reporting added. A huge number of
37 typos fixed and many most probably also left unfixed.

38 **Revision: 03 Date:** 08 August 2004

39 Namespaces added to examples. Long parameter tables slit to smaller tables. Handling missing itemID attributes is
40 now covered (Bug 668). Added note on the use of the static set. When sorting requested, now OK to return unsorted,
41 if sorting not supported by a WSP, but that must be indicated in a response. Changed remaining attribute to integer
42 and defined the use of a special value -1. Added text for special subscription cases, when either starts==expires or
43 duration equals to zero. Added text to cover bug 669 (empty elements not supported well).

44 **Revision: 04 Date:** 23 September 2004

45 Bugs 688, 689 and 690.

46 **Revision: 05 Date:** 27 September 2004

47 Some minor updates as previous updates were not properly completed. Also some typos fixed.

48 **Revision: 2.0-06 Date:** 22 November 2004

49 Update schema to 2.0 (and references).

50 **Contents**

51	1. Overview	5
52	2. Data Model	7
53	3. Message Interface	17
54	4. Querying Data	27
55	5. Modifying Data	43
56	6. Subscriptions and Notifications	50
57	7. The Schema for the DST Protocols	62
58	8. Checklist for Service Specifications	67
59	References	74

60 1. Overview

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

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

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

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

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

82 **Note:**

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

90 1.1. Notation

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

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

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

104 The following namespaces are used in the schema definitions:

- 105 • The prefix `xs:` stands for the W3C XML schema namespace (<http://www.w3.org/2001/XMLSchema>),
106 [\[Schema1\]](#)
- 107 • The prefix `xml:` stands for the W3C XML namespace (<http://www.w3.org/XML/1998/namespace>).
- 108 • The prefix `disco:` stands for the Liberty ID-WSF Discovery Service schema namespace
109 (<urn:liberty:disco:2003-08>). [\[LibertyDisco\]](#)
- 110 • The prefix `md:` stands for the Liberty Metadata schema namespace (<urn:liberty:metadata:2003-08>),
111 [\[LibertyMetadata\]](#)
- 112 • The prefix `sb-ext:` stands for the Liberty ID-WSF SOAP Binding Extension schema namespace
113 (<urn:liberty:sb:2004-04>). [\[LibertySOAPBinding\]](#)

114 The following namespaces are used in examples:

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

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

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

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

124 **1.2. Liberty Considerations**

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

129 2. Data Model

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

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

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

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

147 2.1. Guidelines for Schemas

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

- 150 • Each data attribute regarding the Principal SHOULD be defined as an XML element of a suitable type.
- 151 • XML attributes SHOULD be used only to qualify the data attribute defined as XML elements and not contain the
152 actual data values related to the Principal.
- 153 • An XML element SHOULD either contain other XML elements or actual data value. An XML element SHOULD
154 NOT have mixed content, i.e. both a value and sub-elements. Also complex types all and choice SHOULD
155 NOT be used.
- 156 • Once a data attribute has been published in a specification for a service type, its syntax and semantics MUST not
157 change. If evolution in syntax or semantics is needed, any new version of a data attribute MUST be assigned a
158 different name, effectively creating a new attribute with new semantics so that it does not conflict with the original
159 attribute definition.
- 160 • All elements MUST be defined as global elements, when they can be requested individually. When elements with
161 complex type are defined, references to global elements are used. The reason for this guideline is that the XML
162 Schema for a service does not only define the syntax of the data supported by the service but also the transfer
163 syntax. In many cases it should be possible to query and modify individual elements.
- 164 • The type definitions provided by the XML Schema SHOULD be used, when they cover the requirements.

165 **2.2. Extending a Service**

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

- 168 • An implementation MAY add new elements and attributes to the specified schema. These new elements and
169 attributes MUST use their own XML namespace until they are adopted by the official Liberty specification and
170 schema of the service type.
- 171 • When new features for a service are specified (e.g. new elements), new keywords SHOULD be specified for
172 indicating the new features using the `<Option>` element (see [\[LibertyDisco\]](#) for more information).
- 173 • New values for enumerators MAY be specified subsequent to the release of a specification document for a
174 specific service type. The specification for a service type MUST specify the authority for registering new official
175 enumerators (whether that authority is the specification itself, or some external authority). For specification done
176 by Liberty Alliance, see [\[LibertyReg\]](#).
- 177 • Elements defined in the XML schema for a service type MAY contain an `<xs:any>` element to support ar-
178bitrary schema extension. When the `<xs:any>` elements are in the schema, an implementation MAY sup-
179port this type of extension, but is not required to. The `<xs:any>` elements SHOULD always be put inside
180 `<Extension>` elements. If an implementation does support this type of schema extension, then it MAY regis-
181ter `urn:liberty:dst:can:extend` discovery option keyword. When a service holds new data, which is not
182 defined in the schema for the service type but is stored using this kind of support for extensions, it MAY register
183 `urn:liberty:dst:extend` discovery option keyword.

184 **2.3. Time Values and Synchronization**

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

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

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

193 **2.4. Common Attributes**

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

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

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

207 **2.4.1. The `commonAttributes` Attribute Group**

208 There are only two common attributes:

209 `id` [Optional]

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

219 `modificationTime` [Optional]

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

226 **2.4.2. The `leafAttributes` Attribute Group**

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

228 `modifier` [Optional]

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

231 `ACC` [Optional]

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

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

237 • `urn:liberty:dst:acc:unknown`

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

241 • `urn:liberty:dst:acc:incentive`

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

- 244 • `urn:liberty:dst:acc:challenge`
245 A challenge mechanism has been used to validate the collected data (e.g. an email sent to address and a reply
246 received or an SMS message sent to a mobile phone number containing a WAP URL to be clicked to complete the
247 data collection)
- 248 • `urn:liberty:dst:acc:secondarydocuments`
249 The value has been validated from secondary documents (such as the address from an electric bill).
- 250 • `urn:liberty:dst:acc:primarydocuments`
251 The value has been validated from primary documents (for example, the name and identification number from a
252 passport).
- 253 Other values are allowed for `ACC`, but this specification normatively defines usage only for the values listed
254 above.
255 When the `ACC` is included in the response message, the response SHOULD be signed by the service provider
256 hosting the data service.
- 257 `ACCTime` [Optional]
258 This defines the time that the value for the `ACC` attribute was given. Note that this can be different from the
259 `modificationTime`. The `ACC` contains information that may be related to the validation of the entry. Such
260 validation might happen later than the time the entry was made, or modified. The entry can be validated more
261 than once.

262 2.4.3. The `localizedLeafAttributes` Attribute Group

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

- 265 `xml:lang` [Required]
266 This defines the language used for the value of a localized leaf element. When the
267 `localizedLeafAttributes` attribute group is used for an element, this is a mandatory XML attribute.
- 268 `script` [Optional]
269 Sometimes the language does not define the writing system used. In such cases, this attribute defines
270 the writing system in more detail. This specification defines the following values for this attribute:
271 `urn:liberty:dst:script:kana` and `urn:liberty:dst:script:kanji`. See [\[LibertyReg\]](#) where to
272 find more values and how to specify more values.

273 2.4.4. Individual common attributes

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

- 278 `refreshOnOrAfter`
279 A WSC SHOULD cache the element and its content until the time specified by the `refreshOnOrAfter`
280 attribute, if it wants to use it later and is also allowed to do so. There is no absolute guarantee that the data
281 has not changed before that time, but this is the guidance given by the WSP providing the information.

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

286 2.5. Common Data Types

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

292 The following type definitions are provided:

- 293 • `DSTLocalizedString`
- 294 • `DSTString`
- 295 • `DSTInteger`
- 296 • `DSTURI`
- 297 • `DSTDate`
- 298 • `DSTMonthDay`

299 2.6. The Schema for Common XML Attributes and Data Types

```
300
301     <?xml version="1.0" encoding="UTF-8"?>
302 <xs:schema
303     xmlns:xs="http://www.w3.org/2001/XMLSchema"
304     xmlns:disco="urn:liberty:disco:2004-12"
305     elementFormDefault="qualified"
306     attributeFormDefault="unqualified"
307     xmlns:sb20="urn:liberty:sb:2004-12">
308
309     <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
310     <xs:import namespace="urn:liberty:disco:2004-12" schemaLocation="liberty-idwsf-disco
311 -svc-v2.0.xsd"/>
312     <xs:import namespace="urn:liberty:sb:2004-12" schemaLocation="liberty-idwsf-soap-binding-v2.0.x
313 sd"/>
314
315     <xs:annotation>
316     <xs:documentation>
317 The source code in this XSD file was excerpted verbatim from:
318
319 Liberty ID-WSF Data Services Template Specification
320 Version 2.0-06 Draft
321 22 November 2004
322
323 Copyright (c) 2004 Liberty Alliance participants, see
324 http://www.projectliberty.org/specs/idwsf_copyrights.html
325
326 NOTE: This schema must be used within the context of another schema -
327 It is not intended to validate by itself.
328
```

```

329 The scheme which includes this must provide definitions for:
330 TypeType
331 SelectType
332 TriggerType
333 SortType
334
335     </xs:documentation>
336 </xs:annotation>
337 <xs:element name="ResourceID" type="disco:ResourceIDType"/>
338 <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>
339 <xs:group name="ResourceIDGroup">
340   <xs:choice>
341     <xs:element ref="ResourceID"/>
342     <xs:element ref="EncryptedResourceID"/>
343   </xs:choice>
344 </xs:group>
345 <xs:element name="ChangeFormat">
346   <xs:simpleType>
347     <xs:restriction base="xs:string">
348       <xs:enumeration value="ChangedElements"/>
349       <xs:enumeration value="CurrentElements"/>
350     </xs:restriction>
351   </xs:simpleType>
352 </xs:element>
353 <xs:attribute name="changeFormat">
354   <xs:simpleType>
355     <xs:restriction base="xs:string">
356       <xs:enumeration value="ChangedElements"/>
357       <xs:enumeration value="CurrentElements"/>
358       <xs:enumeration value="All"/>
359     </xs:restriction>
360   </xs:simpleType>
361 </xs:attribute>
362 <!-- Querying Data -->
363 <xs:element name="Query" type="QueryType"/>
364 <xs:complexType name="QueryType">
365   <xs:sequence>
366     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
367     <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded">
368       <xs:complexType>
369         <xs:sequence>
370           <xs:annotation>
371             <xs:documentation>
372               NOTE: The below two types (SelectType and SortType) must
373               be defined by the schema that includes this one.
374             </xs:documentation>
375           </xs:annotation>
376           <xs:element name="Select" type="SelectType" minOccurs="0"/>
377           <xs:element name="Sort" type="SortType" minOccurs="0"/>
378           <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
379         </xs:sequence>
380         <xs:attribute name="id" type="xs:ID"/>
381         <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0"/>
382         <xs:attribute name="itemID" type="IDType"/>
383         <xs:attribute name="changedSince" type="xs:dateTime"/>
384         <xs:attribute name="count" type="xs:nonNegativeInteger"/>
385         <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0"/>
386         <xs:attribute name="setID" type="IDType"/>
387         <xs:attribute name="setReq">
388           <xs:simpleType>
389             <xs:restriction base="xs:string">
390               <xs:enumeration value="Static"/>
391               <xs:enumeration value="DeleteSet"/>
392             </xs:restriction>
393           </xs:simpleType>
394         </xs:attribute>
395       </xs:complexType>

```

```

396         </xs:element>
397         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
398     </xs:sequence>
399     <xs:attribute name="id" type="xs:ID" />
400     <xs:attribute name="itemID" type="IDType" />
401 </xs:complexType>
402
403 <xs:element name="QueryResponse" type="QueryResponseType" />
404 <xs:complexType name="QueryResponseType">
405     <xs:sequence>
406         <xs:element ref="Status" />
407         <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
408             <xs:complexType>
409                 <xs:sequence>
410                     <xs:any minOccurs="0" maxOccurs="unbounded" />
411                 </xs:sequence>
412                 <xs:attribute name="id" type="xs:ID" />
413                 <xs:attribute name="itemIDRef" type="IDReferenceType" />
414                 <xs:attribute name="notSorted" />
415                 <xs:simpleType>
416                     <xs:restriction base="xs:string">
417                         <xs:enumeration value="Now" />
418                         <xs:enumeration value="Never" />
419                     </xs:restriction>
420                 </xs:simpleType>
421                 </xs:attribute>
422                 <xs:attribute ref="changeFormat" />
423                 <xs:attribute name="remaining" type="xs:integer" />
424                 <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0" />
425                 <xs:attribute name="setID" type="IDType" />
426             </xs:complexType>
427         </xs:element>
428         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
429     </xs:sequence>
430     <xs:attribute name="id" type="xs:ID" />
431     <xs:attribute name="itemIDRef" type="IDReferenceType" />
432     <xs:attribute name="timeStamp" type="xs:dateTime" />
433 </xs:complexType>
434 <!-- Modifying Data -->
435 <xs:element name="Modify" type="ModifyType" />
436 <xs:complexType name="ModifyType">
437     <xs:sequence>
438         <xs:group ref="ResourceIDGroup" minOccurs="0" />
439         <xs:element name="Modification" maxOccurs="unbounded">
440             <xs:complexType>
441                 <xs:sequence>
442                     <xs:annotation>
443                         <xs:documentation>
444                             NOTE: The below type must be defined by
445                             the schema that includes this one.
446                         </xs:documentation>
447                     </xs:annotation>
448                     <xs:element name="Select" type="SelectType" minOccurs="0" />
449                     <xs:element name="NewData" minOccurs="0">
450                         <xs:complexType>
451                             <xs:sequence>
452                                 <xs:any minOccurs="0" maxOccurs="unbounded" />
453                             </xs:sequence>
454                         </xs:complexType>
455                     </xs:element>
456                 </xs:sequence>
457                 <xs:attribute name="id" type="xs:ID" />
458                 <xs:attribute name="itemID" type="IDType" />
459                 <xs:attribute name="notChangedSince" type="xs:dateTime" />
460                 <xs:attribute name="overrideAllowed" type="xs:boolean" default="0" />
461             </xs:complexType>
462         </xs:element>

```

```

463     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
464   </xs:sequence>
465   <xs:attribute name="id" type="xs:ID" />
466   <xs:attribute name="itemID" type="IDType" />
467 </xs:complexType>
468 <xs:element name="ModifyResponse" type="ResponseType" />
469 <xs:complexType name="ResponseType">
470   <xs:sequence>
471     <xs:element ref="Status" />
472     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
473   </xs:sequence>
474   <xs:attribute name="id" type="xs:ID" />
475   <xs:attribute name="itemIDRef" type="IDReferenceType" />
476   <xs:attribute name="timeStamp" type="xs:dateTime" />
477 </xs:complexType>
478 <!-- Subscribing notifications and modifying, renewing and deleting existing notifications -->
479 <xs:element name="Subscribe" type="SubscribeType" />
480 <xs:complexType name="SubscribeType">
481   <xs:sequence>
482     <xs:group ref="ResourceIDGroup" minOccurs="0" />
483     <xs:element ref="Subscription" maxOccurs="unbounded" />
484     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
485   </xs:sequence>
486   <xs:attribute name="id" type="xs:ID" />
487   <xs:attribute name="itemID" type="IDType" />
488   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1" />
489 </xs:complexType>
490 <xs:element name="SubscribeResponse" type="SubscribeResponseType" />
491 <xs:complexType name="SubscribeResponseType">
492   <xs:sequence>
493     <xs:element ref="Status" />
494     <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />
495     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
496   </xs:sequence>
497   <xs:attribute name="id" type="xs:ID" />
498   <xs:attribute name="itemIDRef" type="IDReferenceType" />
499   <xs:attribute name="timeStamp" type="xs:dateTime" />
500 </xs:complexType>
501 <!-- Query subscriptions -->
502 <xs:element name="QuerySubscriptions" type="QuerySubscriptionsType" />
503 <xs:complexType name="QuerySubscriptionsType">
504   <xs:sequence>
505     <xs:group ref="ResourceIDGroup" minOccurs="0" />
506     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
507   </xs:sequence>
508   <xs:attribute name="id" type="xs:ID" />
509   <xs:attribute name="itemID" type="IDType" />
510 </xs:complexType>
511 <xs:element name="Subscriptions" type="SubscriptionsType" />
512 <xs:complexType name="SubscriptionsType">
513   <xs:sequence>
514     <xs:element ref="Status" />
515     <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded" />
516     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
517   </xs:sequence>
518   <xs:attribute name="id" type="xs:ID" />
519   <xs:attribute name="itemIDRef" type="IDReferenceType" />
520 </xs:complexType>
521 <!-- Subscription Element -->
522 <xs:element name="Subscription">
523   <xs:complexType>
524     <xs:sequence>
525       <xs:annotation>
526         <xs:documentation>
527           NOTE: The below types (SelectType, TypeType, and TriggerType)
528           must be defined by the schema that includes this one.
529         </xs:documentation>

```

```
530     </xs:annotation>
531     <xs:element name="Select" type="SelectType" minOccurs="0"/>
532     <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
533     <xs:element name="NotifyTo" type="sb20:ServiceInstanceUpdateType"/>
534     <xs:element name="NotifyEndedTo" type="sb20:ServiceInstanceUpdateType" minOccurs="0"/>
535     <xs:element name="Type" type="TypeType" minOccurs="0"/>
536     <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
537     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
538 </xs:sequence>
539 <xs:attribute name="starts" type="xs:dateTime"/>
540 <xs:attribute name="expires" type="xs:dateTime"/>
541 <xs:attribute name="duration" type="xs:duration"/>
542 <xs:attribute name="id" type="xs:ID"/>
543 <xs:attribute name="invokeID" type="IDType"/>
544 <xs:attribute name="subscriptionID" type="IDType"/>
545 <xs:attribute name="includeData" default="Yes">
546   <xs:simpleType>
547     <xs:restriction base="xs:string">
548       <xs:enumeration value="Yes"/>
549       <xs:enumeration value="No"/>
550       <xs:enumeration value="YesWithCommonAttributes"/>
551     </xs:restriction>
552   </xs:simpleType>
553 </xs:attribute>
554 </xs:complexType>
555 </xs:element>
556 <!-- Sending Notifications and Notifying about subscriptions which have ended-->
557 <xs:element name="Notify" type="NotifyEndedType"/>
558 <xs:element name="NotifyResponse" type="NotifyEndedResponseType"/>
559 <xs:element name="Ended" type="NotifyEndedType"/>
560 <xs:element name="EndedResponse" type="NotifyEndedResponseType"/>
561 <xs:complexType name="NotifyEndedType">
562   <xs:sequence>
563     <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
564     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
565   </xs:sequence>
566   <xs:attribute name="id" type="xs:ID"/>
567   <xs:attribute name="itemID" type="IDType"/>
568   <xs:attribute name="timeStamp" type="xs:dateTime"/>
569 </xs:complexType>
570 <xs:complexType name="NotifyEndedResponseType">
571   <xs:sequence>
572     <xs:element ref="Status"/>
573     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
574   </xs:sequence>
575   <xs:attribute name="id" type="xs:ID"/>
576   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
577 </xs:complexType>
578 <!-- Notification Element -->
579 <xs:element name="Notification">
580   <xs:complexType>
581     <xs:sequence>
582       <xs:element name="Data" minOccurs="0">
583         <xs:complexType>
584           <xs:sequence>
585             <xs:any minOccurs="0" maxOccurs="unbounded"/>
586           </xs:sequence>
587         </xs:complexType>
588       </xs:element>
589     </xs:sequence>
590     <xs:attribute name="id" type="xs:ID"/>
591     <xs:attribute name="invokeID" type="IDType"/>
592     <xs:attribute name="subscriptionID" type="IDType" use="required"/>
593     <xs:attribute ref="changeFormat"/>
594     <xs:attribute name="expires" type="xs:dateTime"/>
595     <xs:attribute name="duration" type="xs:duration"/>
596     <xs:attribute name="endReason" type="xs:anyURI"/>

```

```
597         </xs:complexType>
598     </xs:element>
599 </xs:schema>
600
601
```


602 3. Message Interface

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

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

609 **Table 1. Request/Response**

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

610 **Table 2. Notification/Acknowledgement**

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

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

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

622 3.1. Status and Fault Reporting

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

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

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

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

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

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

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

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

649 3.1.1. <Status> element

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

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

- 656 • ActionNotAuthorized
- 657 • ActionNotSupported
- 658 • AllReturned
- 659 • ChangeHistoryNotSupported
- 660 • ChangedSinceReturnsAll
- 661 • DataTooLong
- 662 • ExistsAlready

- 663 • ExtensionNotSupported
- 664 • Failed
- 665 • FormatNotSupported
- 666 • InvalidData
- 667 • InvalidExpires
- 668 • InvalidEndedTo
- 669 • InvalidEndpoint
- 670 • InvalidResourceID
- 671 • InvalidSelect
- 672 • InvalidSetID
- 673 • InvalidSetReq
- 674 • InvalidSort
- 675 • InvalidSubscriptionID
- 676 • MissingCredentials
- 677 • MissingDataElement
- 678 • MissingEndpointElement
- 679 • MissingExpiration
- 680 • MissingItemID
- 681 • MissingNewDataElement
- 682 • MissingNotifyToElement
- 683 • MissingResourceIDElement
- 684 • MissingSecurityMechIDElement
- 685 • MissingSelect
- 686 • MissingSubscriptionID
- 687 • ModifiedSince
- 688 • NewOrExisting
- 689 • NoMoreElements
- 690 • NoMultipleAllowed
- 691 • NoMultipleResources
- 692 • OK

- 693 • `PaginationNotSupported`
- 694 • `Partial`
- 695 • `RequestedPaginationNotSupported`
- 696 • `RequestedSortingNotSupported`
- 697 • `SecurityMechIDNotAccepted`
- 698 • `SetOrNewQuery`
- 699 • `SortNotSupported`
- 700 • `StaticNotSupported`
- 701 • `TimeOut`
- 702 • `TriggerNotSupported`
- 703 • `TypeNotSupported`
- 704 • `UnexpectedError`
- 705 • `UnspecifiedError`

706 The `<Status>` element may contain other `<Status>` elements supplying more detailed return status information.
707 The `code` attribute of the top level `<Status>` element **MUST** contain one of the following values `OK`, `Partial`
708 or `Failed`. The remainder of the values above are used to indicate more detailed return status inside second level
709 `<Status>` element(s).

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

714 Partial
715 The value Partial means that the processing has succeeded only partially and partially failed, e.g. in
716 the processing of a <Query> element some <QueryItem> element has been processed successfully, but
717 the processing of some other <QueryItem> elements has failed. When the value Partial is used for
718 the code attribute of the top level <Status> element, the top level <Status> element MUST have
719 second level <Status> element(s) to indicate the failed part(s) of a [RequestElement] element or a
720 [NotificationElement] element. The processing of the part(s) not referred to by any of the second
721 level <Status> elements MUST have succeeded. A WSP MUST NOT use the value Partial, if it has not
722 processed the whole [RequestElement] element or [NotificationElement] element.
723 A WSP MUST NOT use the value Partial in case of modification requests, when a failed
724 <Modification> element didn't have a valid itemID attribute, i.e. a WSP is not able to indicate
725 the failed <Modification> element. In those cases a WSP MUST use the value Failed and anything
726 changed based on the already processed part MUST be rolled back.
727 A WSP MAY also choose to fail completely another type of [RequestElement], when only a part of it
728 has failed, if the failed part does not have a valid itemID attribute. When ever the top level value Failed
729 is used instead of Partial due to one or more missing itemID attribute(s), the second level status code
730 MissingItemID MUST be used in addition to any other second level status code.
731 In some cases the most descriptive second level status code may not be used as it e.g. might compromise the
732 privacy of a Principal. In those cases, when the second level status code must be used to indicate the failed
733 parts in a case of a partial failure, the value UnspecifiedError MUST be used for the second level status
734 code.

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

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

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

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

755 3.2. Linking with ids

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

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

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

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

771 3.3. Resources

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

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

781 The `ResourceIDGroup` schema is shown below:

```
782  
783  
784     <xs:element name="ResourceID" type="disco:ResourceIDType"/>  
785     <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>  
786     <xs:group name="ResourceIDGroup">  
787         <xs:choice>  
788             <xs:element ref="ResourceID"/>  
789             <xs:element ref="EncryptedResourceID"/>  
790         </xs:choice>  
791     </xs:group>  
792  
793
```

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

798 **Note:**

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

802 3.3.1. Common processing rules

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

- 805 • A receiver of a message (WSP for request messages and WSC for notification messages) MUST support one
806 [RequestElement] element inside a request message and one [NotificationElement] element inside a
807 notification message and SHOULD support multiple. If a receiver supports only one [RequestElement] or
808 [NotificationElement] element inside a request/notification message and the message contains multiple
809 [RequestElement] or [NotificationElement] elements, the processing of the whole message MUST fail
810 and a status code indicating the failure MUST be returned in the response. A more detailed status code with the
811 value NoMultipleResources SHOULD be returned in addition to the top level status code as it is not possible
812 to handle multiple resources in one message. If a WSP supports accessing multiple resources, it MAY register
813 urn:liberty:dst:multipleResources discovery option keyword.
- 814 • If a request message contains multiple [RequestElement] elements or a notification mes-
815 sage contains multiple [NotificationElement] elements, the sender MUST add itemID at-
816 tributes for each [RequestElement]/[NotificationElement] element. The receiver MUST link
817 the [ResponseElement]/<AcknowledgementElement> elements it is sending to corresponding
818 [RequestElement]/[NotificationElement] elements it received using the itemIDRef attributes, if
819 there were itemID attributes in the [RequestElement]/[NotificationElement] elements and there were
820 multiple [RequestElement]/[NotificationElement] elements in the request message.
821 If multiple [RequestElement] or [NotificationElement] elements are used in a message, a receiver
822 MUST discard any [RequestElement] or [NotificationElement] element not having a valid itemID
823 attribute. For that [RequestElement] or [NotificationElement] element no <ResponseElement> or
824 [AcknowledgementElement] element is returned as it is not possible to indicate for which [RequestElement]
825 or [NotificationElement] element that element is returned.
826 The itemIDRef attribute in a [ResponseElement]/<AcknowledgementElement> element MUST have the
827 same value as the itemID attribute in the corresponding [RequestElement]/[NotificationElement] ele-
828 ment.
- 829 • If the processing of a [RequestElement]/[NotificationElement] fails for some reason, any other
830 [RequestElement]/[NotificationElement] elements included in the same message SHOULD be processed
831 normally, as if the error had not occurred as different [RequestElement]/[NotificationElement] elements
832 inside the same message are usually independent. When processing of a [RequestElement]/[NotificationElement]
833 fails completely, the top level status code Failed MUST be used to indicate the failure and a more detailed status
834 code SHOULD be used to indicate more detailed status information. If it is possible and allowed by the service
835 specification to process successfully independent part(s) of the [RequestElement]/[NotificationElement]
836 the top level status code Partial MUST be used as specified earlier. A successful request/notification MUST be
837 indicated using the top level status code OK. Note: even with top level status code OK second level status codes
838 MAY be used.

839 A WSP must know which resource a WSC wants to access to be able to process the query. The following rules apply
840 to resource identifiers:

- 841 • If there is no <ResourceID> or <EncryptedResourceID> element in the [RequestElement], the processing
842 of the whole [RequestElement] MUST fail and a status code indicating failure MUST be returned in the re-
843 sponse, unless the <ResourceID> element would have had the value urn:liberty:isf:implied-resource
844 (see [LibertyDisco]). In this case the <ResourceID> MAY be left out. When either the <ResourceID> or
845 the <EncryptedResourceID> element should have been present, a more detailed status code with the value
846 MissingResourceIDElement SHOULD be used in addition to the top level status code.

847 • If the resource identified in the `<ResourceID>` or `<EncryptedResourceID>` element does not exist, the
848 processing of the whole `[RequestElement]` MUST fail and a status code indicating the failure MUST be
849 returned in the response. A more detailed status code with the value `InvalidResourceID` SHOULD be used
850 in addition to the top level status code.

851 **3.4. `<Select>` element**

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

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

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

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

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

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

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

884 **3.4.1. Common processing rules**

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

886 • If the `<Select>` element is missing from a subelement of a `[RequestElement]` element, when it is supposed to
887 be use, the processing of that subelement MUST fail and a status code indicating the failure MUST be returned
888 in the response. A more detailed status code with the value `MissingSelect` SHOULD be used in addition to
889 the top level status code. The subelements referred here are the `<QueryItem>`, the `<Modification>` and the

890 <Subscriptions>. All these elements are defined later with other protocol elements. Note: in some cases the
891 <Select> element is not needed

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

898 3.5. The timeStamp Attribute

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

903 3.5.1. Common processing rules

904 • A WSP MUST add a `timeStamp` to a `[ResponseElement]`, if the processing of the `[RequestElement]`
905 was successful and a WSP supports either the `changedSince` attribute or the `notChangedSince` attribute or
906 both properly. A WSP MUST also add a `timeStamp` to a `[NotificationElement]`, it supports either the
907 `changedSince` attribute or the `notChangedSince` attribute or both properly. The `timeStamp` attribute MUST
908 have a value which can also be used as a value for the `changedSince` attribute, when querying changes made after
909 the request for which the `timeStamp` was returned or the notification, which had the `timeStamp`. The value of the
910 `timeStamp` attribute MUST also be such that it can be used as a value for the `notChangedSince` attribute, when
911 making modifications after the request for which the `timeStamp` was returned or after receiving the notification
912 message, which carried the `timeStamp` and the modifications will not succeed, if there has been any modification
913 after this request/notification.

914 3.6. The <Extension> Element

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

918 3.7. General error handling

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

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

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

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

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

938 4. Querying Data

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

944 4.1. The <Query> Element

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

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

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

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

```
963 <xs:element name="ChangeFormat">
964   <xs:simpleType>
965     <xs:restriction base="xs:string">
966       <xs:enumeration value="ChangedElements"/>
967       <xs:enumeration value="CurrentElements"/>
968     </xs:restriction>
969   </xs:simpleType>
970 </xs:element>
```

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

974 includeCommonAttributes [Optional]

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

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

981 changedSince [Optional]

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

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

988 In addition to the `id` attribute, the `<QueryItem>` element can have also the `itemID` attribute. This `itemID` attribute
989 is necessary when the `<Query>` element contains multiple `<QueryItem>` elements. The response message can refer
990 to `itemID` attributes of the `<QueryItem>` elements. Also the `<Query>` element can have the `itemID` attribute.
991 `<QueryResponse>` elements in the response message can be mapped to the corresponding `<Query>` elements using
992 this attribute.

993 4.1.1. Pagination

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

998 `count` [Optional]
999 The `count` attribute defines, how many elements should returned in a response. This is the amount of the
1000 elements directly addressed by the `<Select>`, their descendants are automatically included in the response,
1001 if not elsewhere otherwise specified.

1002 `offset` [Optional]
1003 The `offset` attribute specifies, from which element to continue, when querying for more data. The default
1004 value is zero, which refers to the first element.

1005 Changes may happen while a WSC is requesting the data in smaller sets as this requires multiple `<Query>` messages
1006 and so will cause multiple `<QueryResponse>`s. This is not a problem for many services, but with some services
1007 this might cause problems as an inconsistent set of data may be returned to the requesting WSC. If supported by the
1008 service type and the WSP, a WSC may request that the modifications done by others are not allowed to effect what the
1009 requesting WSC gets. In the first `<Query>` of a sequence, the requesting WSC includes the `setReq` attribute with the
1010 value `Static`. The query response returns an identification for the set and in the following queries, this is included as
1011 the value of the `setID` attribute. At the end the WSC requests that the set is deleted (`setReq="DeleteSet"`) to free
1012 the resources on the WSP side.

1013 `setID` [Optional]
1014 The `setID` attribute contains an identification of a set. This must be used by a WSC, when it wants to make
1015 sure that no modifications are done to the set, while it is querying the data from the set.

1016 `setReq` [Optional]
1017 With the `setReq` attribute a WSC is able to request that a consistent set is created for coming queries (value
1018 `Static`) or a set is deleted (`DeleteSet`).

1019 A service specification MUST specify, for which elements the pagination is supported. It is not meant to be available
1020 for each request, just for a selected type of requests. As the use of the static sets consumes more resources on the
1021 server side than normal pagination, the use of static sets must be thought carefully.

1022 4.1.2. The schema for the `<Query>` element

```
1023 <xs:element name="Query" type="QueryType"/>
1024 <xs:complexType name="QueryType">
1025   <xs:sequence>
1026     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
1027     <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded"/>
1028
```

```

1029     <xs:complexType>
1030         <xs:sequence>
1031             <xs:element name="Select" type="SelectType" minOccurs="0"/>
1032             <xs:element name="Sort" type="SortType" minOccurs="0"/>
1033             <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
1034         </xs:sequence>
1035         <xs:attribute name="id" type="xs:ID"/>
1036         <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0"/>
1037         <xs:attribute name="itemID" type="IDType"/>
1038         <xs:attribute name="changedSince" type="xs:dateTime"/>
1039         <xs:attribute name="count" type="xs:nonNegativeInteger"/>
1040         <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0"/>
1041         <xs:attribute name="setID" type="IDType"/>
1042         <xs:attribute name="setReq">
1043             <xs:simpleType>
1044                 <xs:restriction base="xs:string">
1045                     <xs:enumeration value="Static"/>
1046                     <xs:enumeration value="DeleteSet"/>
1047                 </xs:restriction>
1048             </xs:simpleType>
1049         </xs:attribute>
1050     </xs:complexType>
1051 </xs:element>
1052 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1053 </xs:sequence>
1054 <xs:attribute name="id" type="xs:ID"/>
1055 <xs:attribute name="itemID" type="IDType"/>
1056 </xs:complexType>
1057

```

1058 4.2. The <QueryResponse> Element

1059 In addition to different identifiers the <QueryResponse> can contain three different things: requested data elements
1060 with some parameters, a status code and a time stamp.

1061 The requested data is encapsulated inside <Data> elements. One <Data> element contains data requested by one
1062 <QueryItem> element. If there were multiple <QueryItem> elements in the <Query>, the <Data> elements are
1063 linked to their corresponding <QueryItem> elements using the itemIDRef attributes.

1064 If a WSC requested sorting, but a WSP does not support the requested type of sorting or sorting in general, a WSP
1065 SHOULD return the data unsorted, but then it MUST indicate this by including the attribute notSorted within the
1066 <Data> element carrying the unsorted data. The notSorted attribute may have either the value Now, when the
1067 requested sorting is not supported, but sorting in general is, or Never, when the sorting is not supported at all.

1068 If a WSC was querying for changes, the <Data> element may contain the attribute changeFormat to indicate in
1069 which format the changes are returned. The schema for the changeFormat attribute is following:

```

1070
1071     <xs:attribute name="changeFormat">
1072         <xs:simpleType>
1073             <xs:restriction base="xs:string">
1074                 <xs:enumeration value="ChangedElements"/>
1075                 <xs:enumeration value="CurrentElements"/>
1076                 <xs:enumeration value="All"/>
1077             </xs:restriction>
1078         </xs:simpleType>
1079     </xs:attribute>
1080

```

1081 The <Data> element contains also attributes nextOffset and remaining, when a smaller set of the data instead
1082 all the data was requested using the count and the offset attributes in the request. The nextOffset attribute in
1083 a response is the offset of the first item not included in the response. So the value of the nextOffset attribute in
1084 a response can be used directly for the offset attribute in the next request, when the data is fetched sequentially.

1085 The remaining attribute defines, how many items there are after the last item included in the response. The setID
1086 attribute is also included, when a static set is accessed.

1087 If there were multiple <Query> elements in the request message, the <QueryResponse> elements are linked to
1088 corresponding <Query> elements with itemIDRef attributes.

1089 The schema for <QueryResponse> is below:

```
1090
1091 <xs:element name="QueryResponse" type="QueryResponseType"/>
1092 <xs:complexType name="QueryResponseType">
1093   <xs:sequence>
1094     <xs:element ref="Status"/>
1095     <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
1096       <xs:complexType>
1097         <xs:sequence>
1098           <xs:any minOccurs="0" maxOccurs="unbounded"/>
1099         </xs:sequence>
1100         <xs:attribute name="id" type="xs:ID"/>
1101         <xs:attribute name="itemIDRef" type="IDReferenceType"/>
1102         <xs:attribute name="notSorted">
1103           <xs:simpleType>
1104             <xs:restriction base="xs:string">
1105               <xs:enumeration value="Now"/>
1106               <xs:enumeration value="Never"/>
1107             </xs:restriction>
1108           </xs:simpleType>
1109         </xs:attribute>
1110         <xs:attribute ref="changeFormat"/>
1111         <xs:attribute name="remaining" type="xs:integer"/>
1112         <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0"/>
1113         <xs:attribute name="setID" type="IDType"/>
1114       </xs:complexType>
1115     </xs:element>
1116     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1117   </xs:sequence>
1118   <xs:attribute name="id" type="xs:ID"/>
1119   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
1120   <xs:attribute name="timeStamp" type="xs:dateTime"/>
1121 </xs:complexType>
1122
```

1123 4.3. Processing Rules

1124 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

1125 One <Query> element can contain multiple <QueryItem> elements. The following rules specify how those must be
1126 supported and handled:

- 1127 • A WSP MUST support one <QueryItem> element inside a <Query> and SHOULD support multiple. If a WSP
1128 supports only one <QueryItem> element inside a <Query> and the <Query> contains multiple <QueryItem>
1129 elements, the processing of the whole <Query> MUST fail and a status code indicating failure MUST be returned
1130 in the response. A more detailed status code with the value NoMultipleAllowed SHOULD be used in addition
1131 to the top level status code. If a WSP supports multiple <QueryItem> elements inside a <Query>, it MAY register
1132 the urn:liberty:dst:multipleQueryItems discovery option keyword.

- 1133 • If the <Query> contains multiple <QueryItem> elements, the WSC MUST add itemID attributes to each
1134 <QueryItem> element. The WSP MUST link the <Data> elements to corresponding <QueryItem> elements
1135 using the itemIDRef attributes, if there were itemID attributes in the <QueryItem> elements and there were
1136 multiple <QueryItem> elements in the <Query>. The itemIDRef attribute in a <Data> element MUST have
1137 the same value as the itemID attribute in the corresponding <QueryItem> element.
- 1138 • If processing of a <QueryItem> fails, any remaining unprocessed <QueryItem> elements SHOULD NOT be
1139 processed. The data for the already processed <QueryItem> elements SHOULD be returned in the response
1140 message and the status code MUST indicate the failure to completely process the whole <Query>. A more detailed
1141 status SHOULD be used in addition to the top level status code to indicate the reason for failing to process the first
1142 failed <QueryItem>.
- 1143 The following rules specify how the <Select> element should be processed and interpreted:
- 1144 • If there is no changedSince attribute in the <QueryItem> element and the <Select> requests valid data
1145 element(s), but there are no values, the WSP MUST NOT return any <Data> element for that <QueryItem>.
- 1146 • If the <Select> requests multiple data elements, the WSP MUST return all of those data elements inside the
1147 <Data> element corresponding to the containing <QueryItem>.
- 1148 A WSC may request that the data returned is sorted.
- 1149 • When the <Sort> element is included in a <QueryItem> element, the data returned inside a <Data> element
1150 SHOULD be sorted according to the criteria given in the <Sort> element. If a WSP doesn't support sorting,
1151 it SHOULD return the requested data unsorted. When the data is returned unsorted, the notSorted attribute
1152 MUST be used in the <Data> element containing the unsorted data. A WSP MAY also choose to fail to process
1153 the <QueryItem>, if it does not support sorting. In that case the second level status code SortNotSupported
1154 SHOULD be used in addition to the top level status code. A WSP may also register discovery option keyword
1155 urn:liberty:dst:noSorting, if the sorting has been specified for the service type, but the WSP doesn't
1156 support it.
- 1157 • If the content of the <Sort> element is not according to service specifications, a WSP SHOULD return the
1158 requested data unsorted. When the data is returned unsorted, the notSorted attribute MUST be used in the
1159 <Data> element containing the unsorted data and the second level status code InvalidSort SHOULD also be
1160 used. A WSP MAY also choose to fail to process the <QueryItem>, if the content of the <Sort> element is not
1161 according to service specifications. In this kind of a case the second level status code InvalidSort SHOULD be
1162 used in addition to the top level status code. If the the content of the <Sort> element is valid, but a WSP does not
1163 support the requested type of sorting, it SHOULD return the requested data unsorted. When the data is returned
1164 unsorted, the notSorted attribute MUST be used in the <Data> element containing the unsorted data. A WSP
1165 MAY also choose to fail to process of the <QueryItem>, if it does not support the requested type of sorting. It
1166 SHOULD use the second level status code RequestedSortingNotSupported in addition to the top level status
1167 code.
- 1168 • When the notSorted attribute is used, it MUST have the value Now, when a WSP supports sorting, but not the
1169 requested type or the content of the <Sort> element was invalid. The notSorted attribute MUST have the value
1170 Never, when a WSP does not support sorting at all.
- 1171 A WSC may want to receive the data in smaller sets instead of getting all the data at once, when there can be many
1172 elements with the same name. A WSC indicates this using either or both of the attributes count and offset in a
1173 <QueryItem> element, when the <Select> addresses a set of elements all having the same name. The number of
1174 elements inside this set may be restricted further by other parameters. Also access rights and policies may reduce the
1175 set of elements a WSC is allowed to get.

- 1176 • A WSP MUST always follow the same ordering procedure, when the <Select> and <Sort> elements have the
1177 same values and either or both of attributes `count` and `offset` are used in the <QueryItem> element. This is
1178 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
1179 previously returned items with five new items.
- 1180 • When either or both of the attributes `count` and `offset` is used in a <QueryItem> element and a WSP doesn't
1181 support pagination, the processing of whole <QueryItem> element MUST fail and the second level status code
1182 `PaginationNotSupported` SHOULD be used in addition to the top level status code. A WSP may support
1183 pagination, but not for the requested elements. In such a case the processing of whole <QueryItem> element
1184 MUST fail and the second level status code `RequestedPaginationNotSupported` SHOULD be used in
1185 addition to the top level status code. If a WSP doesn't support pagination at all, it MAY register the discovery
1186 option keyword `urn:liberty:dst:noPagination` to indicate this.
- 1187 • When the `count` attribute is included in a <QueryItem> element, the corresponding <Data> element in the
1188 <QueryResponse> MUST NOT contain more elements addressed with the value of the <Select> element than
1189 specified by the `count` attribute. A WSP MAY return a smaller number of elements of the same name that
1190 requested by a WSC. If the `count` attribute has the value zero, the WSP MUST NOT return any data elements
1191 inside the <Data> element. This `count="0"` may be used for querying the number of remaining elements starting
1192 from the specified offset, e.g. from offset zero, i.e. the total number of the elements addressed by the <Select>
1193 element. When the `count` attribute is not used in a <QueryItem> element, it means that the WSC requests for
1194 all data specified by other parameters like the <Select> element starting from the specified offset. As the default
1195 value for the `offset` attribute is zero, the case when neither of the attributes `offset` or `count` is not present
1196 reduces to a normal query.
- 1197 • When pagination is requested by a WSC, the elements inside a <Data> element MUST be in the ascending order
1198 of their offsets. The first element MUST have the offset specified by the `offset` attribute in the <QueryItem>
1199 element. The <Data> element MUST have both attributes `nextOffset` and `remaining`. The `nextOffset`
1200 attribute MUST have the offset of the first element not returned in the response. The value of the `remaining`
1201 attribute MUST define how many elements there are left starting from the value of the `nextOffset`, if a WSP
1202 knows that (e.g. that information might not be available from a backend system). If WSP does not know the
1203 exact value, it MUST use the value -1 for the `remaining` attribute until it knows the value or there is no data left
1204 (`remaining="0"`). When `remaining="-1"`, a WSC must make new requests until `remaining="0"`, if it wants
1205 to get all the data.
- 1206 • When the `setReq` attribute is included in a <QueryItem> element and has the value `Static`, the WSP SHOULD
1207 return the `setID` attribute to the requesting WSC and process <QueryItem> elements later having this `setID`
1208 based on the data the WSP has at the time, when the value for the `setID` was created. If a WSP receives a
1209 <QueryItem> element having the `setReq` attribute and does not support static sets for the requested data or not at
1210 all, the processing of the <QueryItem> element MUST fail and a second level status code `StaticNotSupported`
1211 SHOULD be used in addition to the top level status code. If a WSP doesn't support static sets at all, it MAY register
1212 the discovery option keyword `urn:liberty:dst:noStatic`.
- 1213 • When the `setID` attribute is included in a request, the following parameters MUST NOT be used in a
1214 <QueryItem> element: the <Select> element, the <Sort> element, the `changedSince` attribute or the
1215 `includeCommonAttributes` attribute. The requests are made from an earlier defined static set and the `count`
1216 and the `offset` attributes are used to define, what is requested from that set. If any of the mentioned parameters
1217 is present, when the `setID` attribute is used, it is unclear what a WSC wants and the processing of the whole
1218 <QueryItem> MUST fail and a second level status code `SetOrNewQuery` SHOULD be used in addition to the
1219 top level status code.
- 1220 • When the `setID` attribute is included in a <QueryItem> element and has a valid value, the <Data> element in
1221 the response MUST always have the `setID` attribute.

- 1222 • When a static set is created, the requesting WSC MUST query all the data it needs from this set as soon as possible
1223 and delete the static set immediately after this using `setReq="DeleteSet"`. A WSP MAY also delete the static
1224 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
1225 non-existing static set, the processing of the whole `<QueryItem>` MUST fail and the second level status code
1226 `InvalidSetID` SHOULD be used in addition to the top level status code.
- 1227 • The `setReq="Static"` and the `setID` attribute MUST NOT be used simultaneously in a `<QueryItem>` element.
1228 If they are used, the WSP MUST ignore the `setReq="Static"` and process the `<QueryItem>` element like the
1229 `setReq` attribute would not be present.
- 1230 • If the `setReq` attribute has some other value than `Static` or `DeleteSet`, the processing of the whole
1231 `<QueryItem>` element must fail and a second level status code `InvalidSetReq` SHOULD be used in addition
1232 to the top level status code.
- 1233 Even when the requested data exists, it should be noted that access and privacy policies specified by the resource owner
1234 may cause the request to result in data not being returned to the requester.
- 1235 • When a WSP processes a `<QueryItem>`, it MUST check whether the resource owner (the Principal, for example)
1236 has given consent to return the requested information. To be able to check WSC specific access rights, the WSP
1237 MUST authenticate the WSC (see [\[LibertySecMech\]](#) and [\[LibertyMetadata\]](#)). The WSP MUST also check that
1238 any usage directive given in the request is acceptable based on the usage directives defined by the resource owner
1239 (see [\[LibertySOAPBinding\]](#)). If either check fails for any piece of the requested data, the WSP MUST NOT return
1240 that piece of data. Note that there can be consent for returning some data element, but not its attributes. E.g. a
1241 resource owner might not want to release the `modifier` attribute, if she does not want to reveal information about
1242 which services she uses. The data for which there is no consent from the resource owner MUST be handled as
1243 if there was no data. The WSP MAY try to get consent from the resource owner while processing the request,
1244 e.g. by using an interaction service (see [\[LibertyInteract\]](#)). A WSP might check the access rights and policies
1245 in usage directives at a higher level, before getting to DST processing and MAY, in this case, just return an ID-*
1246 Fault Message [\[LibertySOAPBinding\]](#) without processing the `<Query>` element at all, if the requesting WSC is
1247 not allowed to access data.
- 1248 It is possible to query changes since a specified time using the `changedSince` attribute. The following rules specify
1249 how this works:
- 1250 • If the `<QueryItem>` element contains the `changedSince` attribute, the WSP SHOULD return only those elements
1251 addressed by the `<Select>` which have been modified since the time specified in the `changedSince` attribute.
1252 There are two different formats, in which the changed data can be returned. A WSC SHOULD indicate using
1253 the `<ChangeFormat>` element the format it prefers and also, if it understands the other format. The two formats
1254 are `ChangedElements` and `CurrentElements`. If a service specification doesn't specify anything else the value
1255 `ChangedElements` MUST be used as a default value as it is compatible with the format used in the version 1.0
1256 of the Data Services Template.
- 1257 • A WSP MUST ignore the `<ChangeFormat>` element, if the `changedSince` attribute is not used in the same
1258 `<QueryItem>` element. A WSP MUST NOT use a format, which a WSC does not understand. Note that format
1259 `ChangedElements`, has the format `All` as a fallback solution, when a WSP doesn't have all the needed change
1260 history information. Also if a WSP doesn't support requesting only changed data, it returns all data.
- 1261 • A `<QueryItem>` element MAY contain two `<ChangeFormat>` element with different values. A WSP SHOULD
1262 use the format specified by the first `<ChangeFormat>` element, but, if it does not support that format, it MAY use
1263 the format specified by the second `<ChangeFormat>` element.

- 1264 • If a WSP does not support the format a WSC is requesting to be used, the processing of the `<QueryItem>` MUST
1265 fail and the second level status code `FormatNotSupported` SHOULD be used in addition to the top level status
1266 code.
- 1267 • If a WSC requests the `ChangedElements` format and a WSP supports it, the WSP SHOULD return only the
1268 changed information. If some element has been deleted, a WSP SHOULD return an empty element to indicate the
1269 deletion (`<ElementName/>`). The only allowed exception to this is that the WSP does not have enough history
1270 information available to be able to return only the changed information. In that case it MUST use format `All` and
1271 return all current elements with their values even if those have not changed since the specified time.
- 1272 • If a WSC requests the `CurrentElements` format and a WSP supports it, the WSP SHOULD return only the
1273 currently existing elements. It SHOULD return an empty element, if an element has not changed to indicate that
1274 no change has happened (`<ElementName/>`).
- 1275 • Note: as empty elements are used to indicate either deleted or not changed elements depending on the used format,
1276 the formats `CurrentElements` and `ChangedElements` do not work well, if the data hosted by a service may
1277 contain empty elements. In those cases a service should either use only format `All` or always have some attribute(s)
1278 for the otherwise empty elements.
- 1279 • If a WSC has used the `<ChangeFormat>` element in a request, a WSP MUST use the `changeFormat` attribute
1280 in the response to indicate, which format is used. A WSP MUST not use the `changeFormat` attribute in
1281 a response, if the `<ChangeFormat>` element was not used in the corresponding request so the processing stays
1282 version 1.0 compatible, when the `<ChangeFormat>` element is not used.
- 1283 • If there can be multiple elements with same name, the `id` attribute or some other attribute used to distinguish
1284 the elements from each other MUST be included (e.g. in case of an ID-SIS Personal Profile service the
1285 following empty element could be returned `<AddressCard id="tr7632q"/>` to indicate a deleted or not
1286 changed `<AddressCard>` depending on the used format). If the value of the `id` attribute or some other attribute
1287 used for distinguishing elements with same name is changed, the WSP MUST consider this as a case, in which the
1288 element with the original value of the distinguishing attribute is deleted and a new one with the new value of the
1289 distinguishing attribute is created. To avoid this, a WSP MAY refuse to accept modifications of a distinguishing
1290 attribute and MAY require that an explicit deletion of the element is done and a new one created.
- 1291 • If the elements addressed by the `<Select>` have some values, but there has been no changes since the time
1292 specified in the `changedSince` attribute, the WSP MUST return empty `<Data>` element (`<Data/>`), when it
1293 returns the changes properly. This empty `<Data>` element indicates that no changes have occurred. There might
1294 be cases in which the WSP is not able to return changes properly, see later processing rules. Please note that in
1295 cases that have no values, no `<Data>` element is returned to indicate this. So empty `<Data>` element has different
1296 semantics than missing `<Data>` element.
- 1297 • If the `<QueryItem>` element contains the `changedSince` attribute and a WSP is not keeping track of modification
1298 times, it SHOULD process the `<QueryItem>` element as there would be no `changedSince` attribute, and indicate
1299 this in the response using the second level status code `ChangedSinceReturnsAll`. This is not considered a
1300 failure and the rest of the `<QueryItem>` elements MUST be processed. Also it might be that a WSP does not
1301 have a full change history and so for some queries, it is not possible to find out, which changes occurred after
1302 the specified time. As processing with access rights and policy in place might be quite complex, a WSP might
1303 sometimes process the query for changes properly and sometime process it as if there were no `changedSince`
1304 attribute. In those cases, when a WSP returns all current values, it SHOULD indicate this with the second level
1305 status code `AllReturned` and, if the `<ChangeFormat>` element was used in the request, the `changeFormat`
1306 attribute with the value `All` SHOULD be used. This is also not considered a failure and the rest of the
1307 `<QueryItem>` elements MUST be processed. Please note that the status code `AllReturned` differs from the
1308 status code `ChangedSinceReturnsAll`, as `ChangedSinceReturnsAll` means that the WSP never processes
1309 the `changedSince` attribute properly. A WSP MUST use either `AllReturned` or `ChangedSinceReturnsAll`
1310 as the second level status code, when it returns data, but does not process the `changedSince` attribute properly, i.e.
1311 returns only the changes. If a WSP will not process the `<QueryItem>` elements with a `changedSince` attribute

1312 at all, it MUST indicate this with top level status code `Failed` and SHOULD also return a second level status code
1313 of `ChangeHistoryNotSupported` in the response. In this case a WSP MUST NOT return any `<Data>` element
1314 for the `<QueryItem>` element containing the `changedSince` attribute. If a WSP processes the `changedSince`
1315 attribute, it MUST also support the `notChangedSince` attribute for `<Modification>` element and MAY register
1316 the `urn:liberty:dst:changeHistorySupported` discovery option keyword. Please note that still in some
1317 cases a WSP MAY return `AllReturned`.

1318 • Access rights and policies in place may affect how the queries for changes can work as they affect which elements
1319 and attributes a WSC is allowed to see. If a WSC was originally allowed to get the requested data, but is no longer
1320 after some change in access policies, then from its point of view that data is deleted and that should be taken into
1321 account in the response. If the WSP notices that access rights have changed, and the current rights do not allow
1322 access, it MUST return all data except the data for which the access rights were revoked, and use the second level
1323 status code `AllReturned` and, if the `<ChangeFormat>` element was used in the request, the `changeFormat`
1324 attribute with the value `All` SHOULD be used. The WSP MUST NOT return empty elements for the data for
1325 which access rights were changed even if the format `ChangedElement` was requested, as this might reveal the
1326 fact that this specific data has at least existed at the service in some point of time. Please note that it might be the
1327 case that the data was added after the WSCs access rights were revoked and the WSC was never supposed to be
1328 aware of the existence of that data. If the WSP notices that the access rights are changed and the current rights do
1329 allow access, it MUST consider the data for which the access rights are changed, as if it were just created.

1330 • Both the WSC and WSP may have policies specified by the Principal for control of their data. Only by
1331 comparing policy statements made by the WSC (via `<UsageDirective>` elements (see [\[LibertySOAPBinding\]](#))
1332 with policies maintained on behalf of the Principal by the WSP it is possible to fully determine the effects of
1333 interaction between these sets of policies. As it might be too expensive to search for policies the WSC promised
1334 to honour, when it made the original request, and this information might not even be available, the WSP might be
1335 only capable of making the decision based on the policy changes made by the Principal. If some data is prevented
1336 from being returned to the WSC due to conflicts in policies and the WSP notices that the Principal's policies have
1337 changed, it MUST return all data except that for which the Principal's policy has denied access against the current
1338 policy of a requesting WSC, and use the second level status code `AllReturned` to indicate that the WSC must
1339 check the response carefully to find out what has changed. Also if the `<ChangeFormat>` element was used in
1340 the request, the `changeFormat` attribute with the value `All` SHOULD be used. The WSP MUST NOT return
1341 empty elements for the data for which the Principal's policy was changed even if the format `ChangedElements`
1342 was requested, as this might reveal the fact that this specific data was exposed by the service at some point in time.
1343 Please note that it might be the case that that data has been added after the policies were changed and the requesting
1344 WSC was never supposed to be aware of that data, unless it changed the policy it promises to honour. If the WSP
1345 notices that the Principal's policy has changed and the current policy does allow access, it MUST consider the data
1346 for which the policy is changed as if it had been just created. If a WSC changes the policy it promises to honour,
1347 it SHOULD make a new query without a `changedSince` attribute.

1348 • As mentioned earlier the WSP might in some cases return all the current data the `<Select>` points to, and not
1349 just the changes using specified format, even when the `changedSince` attribute is present. So the WSC MUST
1350 compare the returned data to previous data it had queried earlier to find out what really has changed. Please note
1351 that this MUST be done even when the WSP has processed the `changedSince` correctly, because some values
1352 might have been changed back and forth and now they have same values that they used to have earlier, despite the
1353 most current previous values being different.

1354 The common attributes are not always returned. A WSC may indicate with the `includeCommonAttributes`
1355 attribute, whether it wants to have the common attributes or not.

1356 • If the `includeCommonAttributes` is set to `True`, the common attributes specified by attribute
1357 groups `commonAttributes` and `leafAttributes` MUST be included in the response, if their val-
1358 ues are specified for the requested data elements. The ACC attributes MAY be left out, if the value is
1359 `urn:liberty:dst:acc:unknown`.

- 1360 • If the `id` attribute is used for distinguishing similar elements from each other by the service, it **MUST** be returned,
1361 even if the `includeCommonAttributes` is false. Also, when either or both of the attributes `xml:lang` and
1362 `script` are present, they **MUST** be returned, even if the `includeCommonAttributes` is false

1363 4.4. Examples

1364 Please note that as the first examples are based on the [\[LibertyIDPP\]](#) all elements defined in this specifications are also
1365 in the namespace defined in [\[LibertyIDPP\]](#).

1366 The following query example requests the common name and home address of a Principal:

```
1367
1368 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1369   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1370   <pp:QueryItem itemID="name">
1371     <pp:Select>/pp:PP/pp:CommonName</pp:Select>
1372   </pp:QueryItem>
1373   <pp:QueryItem itemID="home">
1374     <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType="urn:liberty:id-sis-pp:addrType:
1375 home"]</pp:Select>
1376   </pp:QueryItem>
1377 </pp:Query>
1378
```

1379 This query may generate the following response:

```
1380
1381 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1382   <pp:Status code="OK"/>
1383   <pp:Data itemIDRef="name">
1384     <pp:CommonName>
1385       <pp:CN>Zita Lopes</pp:CN>
1386       <pp:AnalyzedName nameScheme="firstlast">
1387         <pp:FN>Zita</pp:FN>
1388         <pp:SN>Lopes</pp:SN>
1389         <pp:PersonalTitle>Dr.</pp:PersonalTitle>
1390       </pp:AnalyzedName>
1391       <pp:AltCN>Maria Lopes</pp:AltCN>
1392       <pp:AltCN>Zita Ma Lopes</pp:AltCN>
1393     </pp:CommonName>
1394   </pp:Data>
1395   <pp:Data itemIDRef="home">
1396     <pp:AddressCard id='9812'>
1397       <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1398       <pp:Address>
1399         <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way North</pp:PostalAddress>
1400         <pp:PostalCode>98503-2341</pp:PostalCode>
1401         <pp:L>Olympia</pp:L>
1402         <pp:ST>wa</pp:ST>
1403         <pp:C>us</pp:C>
1404       </pp:Address>
1405     </pp:AddressCard>
1406   </pp:Data>
1407 </pp:QueryResponse>
1408
```

1409 If there was no user consent for the release of the `<pp:CommonName>` or for the whole `<pp:AddressCard>` with
1410 `pp:AddressType='urn:liberty:id-sis-pp:addrType:home'`, apart from the country information, then the
1411 response is as follows (including a timestamp, as this service supports change history).

```
1412
1413 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1414   timeStamp="2003-02-28T12:10:12Z">
```

```
1415 <pp:Status code="OK"/>
1416 <pp:Data itemIDRef="home">
1417 <pp:AddressCard id='9812'>
1418 <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1419 <pp:Address>
1420 <pp:C>us</pp:C>
1421 </pp:Address>
1422 </pp:AddressCard>
1423 </pp:Data>
1424 </pp:QueryResponse>
1425
```

1426 If there was no <pp:CommonName> and no <pp:AddressCard> with pp:AddressType =
1427 'urn:liberty:id-sis-pp:addrType:home', then the response is:

```
1428
1429
1430 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1431 timeStamp="2003-02-28T12:10:12Z">
1432 <pp:Status code="OK"/>
1433 </pp:QueryResponse>
1434
1435
```

1436 The following request queries the fiscal identification number of the Principal with the common attributes:

```
1437
1438
1439 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1440 <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1441 <pp:QueryItem includeCommonAttributes="True">
1442 <pp:Select>/pp:PP/pp:LegalIdentity/pp:VAT</pp:Select>
1443 </pp:QueryItem>
1444 </pp:Query>
1445
1446
```

1447 This query may generate the following response:

```
1448
1449
1450 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1451 id="12345" timeStamp="2003-05-28T23:10:12Z">
1452 <pp:Status code="OK"/>
1453 <pp:Data>
1454 <pp:VAT modifier="http://www.accountingservices.com"
1455 modificationTime="2003-04-25T15:42:11Z"
1456 attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">
1457 <pp:IDValue modifier="http://www.accountingservices.com"
1458 modificationTime="2003-04-25T15:42:11Z"
1459 attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">50267
1460 7123</IDValue>
1461 <pp:IDType modifier="http://www.accountingservices.com"
1462 modificationTime="2003-03-12T09:12:09Z"
1463 attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">urn:liberty:altIDType:itcif</IDType>
1464 </pp:VAT>
1465 </pp:Data>
1466 </pp:QueryResponse>
1467 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
1468 . . .
1469 . . .
1470 </ds:Signature>
1471
1472
```

1473 The following request queries for address information which has been changed since 12:10:12 28 February 2003 UTC:

```
1474
1475
1476 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1477   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1478   <pp:QueryItem changedSince="2003-02-28T12:10:12Z">
1479     <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
1480   </pp:QueryItem>
1481 </pp:Query>
1482
1483
```

1484 This query can generate following response:

```
1485
1486
1487 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1488 timeStamp="2003-05-30T16:10:12Z">
1489   <pp:Status code="OK"/>
1490   <pp:Data>
1491     <pp:AddressCard id='9812'>
1492       <pp:Address>
1493         <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
1494       </pp:Address>
1495     </pp:AddressCard>
1496     <pp:AddressCard id='w1q2' />
1497   </pp:Data>
1498 </pp:QueryResponse>
1499
1500
```

1501 Please note that only the changed information inside the <pp:AddressCard> is returned. The response shows that
1502 after the specified time, there was also another <pp:AddressCard> present, but that has been deleted. As there can
1503 be many <pp:AddressCard> elements, the id attribute is returned to distinguish distinct elements.

1504 If there have been no changes since the specified time, then the response is just:

```
1505
1506 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1507 timeStamp="2003-05-30T16:10:12Z">
1508   <pp:Status code="OK"/>
1509   <pp:Data/>
1510 </pp:QueryResponse>
1511
```

1512 If the same request for changed addresses is made including the <pp:ChangeFormat> element:

```
1513
1514
1515 <pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1516   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
1517   <pp:QueryItem changedSince="2003-02-28T12:10:12Z">
1518     <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
1519     <pp:ChangeFormat>CurrentElements</pp:ChangeFormat>
1520   </pp:QueryItem>
1521 </pp:Query>
1522
1523
```

1524 All the current elements are returned in the response:

```
1525
1526
1527 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1528 timeStamp="2003-05-30T16:10:12Z">
1529   <pp:Status code="OK"/>
```

```
1530 <pp:Data changeFormat="CurrentElements">
1531   <pp:AddressCard id='9812'>
1532     <pp:Address>
1533       <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
1534       <pp:PostalCode/>
1535       <pp:L/>
1536       <pp:ST/>
1537       <pp:C/>
1538     </pp:Address>
1539   </pp:AddressCard>
1540 </pp:Data>
1541 </pp:QueryResponse>
1542
1543
```

1544 Please note that now all the current elements inside the `<pp:AddressCard>` are returned. The deleted
1545 `<pp:AddressCard>` is not shown at all and for the elements, which have not changed only empty elements
1546 are returned.

1547 If a WSP does not support change history, then the response could be:

```
1548
1549 <pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
1550 timeStamp="2003-05-30T16:10:12Z">
1551   <pp:Status code="OK">
1552     <Status code="ChangeSinceReturnsAll"/>
1553   </pp:Status>
1554   <pp:Data changeFormat="All">
1555     <pp:AddressCard id='9812'>
1556       <pp:AddressType>urn:liberty:id-sis-pp:addrType:home </pp:AddressType>
1557       <pp:Address>
1558         <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
1559         <pp:PostalCode>98503-2341</pp:PostalCode>
1560         <pp:L>Olympia</pp:L>
1561         <pp:ST>wa</pp:ST>
1562         <pp:C>us</pp:C>
1563       </pp:Address>
1564     </pp:AddressCard>
1565   </pp:Data>
1566 </pp:QueryResponse>
1567
```

1568 The rest of the examples are related to pagination and sorting based on fictional address service, so all the DST
1569 elements in the namespace of that fictional address service.

1570 Parameters `<Select>` and `<Sort>` and returned `<Data>` elements do not have valid contents in the examples as the
1571 main point is to show the principle how pagination works and the use of the pagination related attributes

1572 A Resource contains 40 address cards and the WSC A wants to list those ordered by the City and 10 at the time. Due
1573 to access rights and policies the WSC A is allowed to get only 30 of those AddressCards. The WSC A makes the first
1574 query:

```
1575
1576 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1577   <ads:ResourceID>http://provider.example.com/ohj243hj24</ads:ResourceID>
1578   <ads:QueryItem count="10">
1579     <ads:Select>Pointing to the AddressCards</ads:Select>
1580     <ads:Sort>Requesting sorting by the City</ads:Sort>
1581   </ads:QueryItem>
1582 </ads:Query>
1583
1584
```

1585 and gets back the first ten address cards ordered by the City:

```
1586
1587 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1588 timeStamp="2004-03-23T03:40:00Z">
1589   <ads:Status code="OK"/>
1590   <ads:Data remaining="20" nextOffset="10">first ten address cards</ads:Data>
1591 </ads:QueryResponse>
1592
```

1593 Then it queries the next ten starting from offset 10:

```
1594
1595 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1596   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
1597   <ads:QueryItem count="10" offset="10">
1598     <ads:Select>Pointing to the AddressCards</ads:Select>
1599     <ads:Sort>Requesting sorting by the City</ads:Sort>
1600   </ads:QueryItem>
1601 </ads:Query>
1602
```

1603 and gets those

```
1604
1605 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1606 timeStamp="2004-03-23T03:40:20Z">
1607   <ads:Status code="OK"/>
1608   <ads:Data remaining="10" nextOffset="20">next ten address cards</ads:Data>
1609 </ads:QueryResponse>
1610
```

1611 After this the WSC B adds one more address card to the resource. The WSC A is allowed to get this address card, but
1612 when sorting based on the City, this new card has the offset 15. When the WSC A fetches the next ten address cards:

```
1613
1614 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1615   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
1616   <ads:QueryItem count="10" offset="20">
1617     <ads:Select>Pointing to the AddressCards</ads:Select>
1618     <ads:Sort>Requesting sorting by the City</ads:Sort>
1619   </ads:QueryItem>
1620 </ads:Query>
1621
```

1622 It gets ten address cards, but it has already received the first address card already in the previous response.

```
1623
1624 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1625 timeStamp="2004-03-23T03:41:00Z">
1626   <ads:Status code="OK"/>
1627   <ads:Data remaining="1" nextOffset="30">next ten address cards</ads:Data>
1628 </ads:QueryResponse>
1629
```

1630 Finally the WSC A fetches the last address card.

```
1631
1632 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1633   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
1634   <ads:QueryItem count="1" offset="30">
1635     <ads:Select>Pointing to the AddressCards</ads:Select>
1636     <ads:Sort>Requesting sorting by the City</ads:Sort>
1637   </ads:QueryItem>
1638 </ads:Query>
1639
```


1640 and gets the 31st address card from offset 30.

```
1641  
1642 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1643 timeStamp="2004-03-23T03:41:17Z">  
1644   <ads:Status code="OK"/>  
1645   <ads:Data remaining="0" nextOffset="31">the last address card</ads:Data>  
1646 </ads:QueryResponse>  
1647
```

1648 So the WSC A didn't get this new address card added by the WSC B and got one card twice.

1649 In an alternative scenario, if supported by the WSP, the WSC A requests a static set to be created so that simultaneous
1650 modifications can not affect the results the WSC A gets. The initial request includes the `setReq` attribute:

```
1651  
1652 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1653   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>  
1654   <ads:QueryItem count="10" setReq="Static">  
1655     <ads:Select>Pointing to the AddressCards</ads:Select>  
1656     <ads:Sort>Requesting sorting by the City</ads:Sort>  
1657   </ads:QueryItem>  
1658 </ads:Query>  
1659
```

1660 In the response the first ten address cards are returned together with a handle to this static set (the attribute `setID`).

```
1661  
1662 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1663 timeStamp="2004-03-23T03:40:00Z">  
1664   <ads:Status code="OK"/>  
1665   <ads:Data remaining="20" nextOffset="10" setID="gfkjds98">first ten address cards</ads:Data>  
1666 </ads:QueryResponse>  
1667
```

1668 In the next query the WSC A queries the next ten address card referring to the static set using the `setID` attribute. The
1669 `<Select>` element is not anymore used.

```
1670  
1671 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1672   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>  
1673   <ads:QueryItem count="10" offset="10" setID="gfkjds98"/>  
1674 </ads:Query>  
1675
```

1676 In the response the next ten address cards are returned and the `setID` is still returned as always when accessing a static
1677 set.

```
1678  
1679 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"  
1680 timeStamp="2004-03-23T03:40:00Z">  
1681   <ads:Status code="OK"/>  
1682   <ads:Data remaining="10" nextOffset="20" setID="gfkjds98">next ten address cards</ads:Data>  
1683 </ads:QueryResponse>  
1684
```

1685 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
1686 ten address cards.

```
1687  
1688 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">  
1689   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>  
1690   <ads:QueryItem count="10" offset="20" setID="gfkjds98"/>  
1691 </ads:Query>  
1692
```

1693 So the WSC A gets the last ten address card.

```
1694
1695 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1696 timeStamp="2004-03-23T03:40:00Z">
1697   <ads:Status code="OK"/>
1698   <ads:Data remaining="0" nextOffset="30" setID="gfkjds98">next ten address cards</ads:Data>
1699 </ads:QueryResponse>
1700
```

1701 Finally the WSC A deletes the static set. This deletion could have been done together with the previous request, but
1702 the WSC wanted to play safe and delete the static set only after getting all the data it wanted.

```
1703
1704 <ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
1705   <ads:ResourceID>http://provider.com/ohj243hj24</ResourceID>
1706   <ads:QueryItem count="0" setID="gfkjds98" setReq="DeleteSet"/>
1707 </ads:Query>
1708
```

1709 And the WSP acknowledges the request.

```
1710
1711 <ads:QueryResponse xmlns:ads="http://www.example.com/2010/12/AddressService"
1712 timeStamp="2004-03-23T03:40:00Z">
1713   <ads:Status code="OK"/>
1714 </ads:QueryResponse>
1715
```

1716 So the addition the WSC B tried to make is not visible in the static set. Either the WSP refused to accept the addition
1717 while WSC A was accessing the data or it created a temporary set for the WSC A to access and the modification by the
1718 WSC B was accepted, but not visible in the temporary static set created for WSC A. In the example above the WSP
1719 created a temporary set and so returned the same time stamp in all responses containing data from that temporary set.

1720 5. Modifying Data

1721 The data stored by a data service can be given initial values, existing values can be replaced with new values and the
 1722 data can also be removed. Usually the Principal can make these modifications directly at the data service using the
 1723 provided user interface, but these modifications may also be made by other service providers. The <Modify> element
 1724 supports all these operations for service providers which want to modify the data store in data services.

1725 5.1. <Modify> element

1726 The <Modify> element has two sub-elements. Either the <ResourceID> or <EncryptedResourceID> element is
 1727 used to identify the resource which is modified by this request. The <Modification> element specifies which data
 1728 elements of the specified resource should be modified and how. There can be multiple <Modification> elements in
 1729 one <Modify>.

1730 The <Select> element inside a <Modification> element specifies the data this modification should affect. If the
 1731 <Select> element is not used, it means that the whole content of the Resource should be modified. In addition to this
 1732 <Select> element the other main part of the <Modification> element is the <NewData> element. The <NewData>
 1733 element defines the new values for the data addressed by the <Select> element. The new values specified inside the
 1734 <NewData> element replace existing data, if the overrideAllowed attribute of the <Modification> element is
 1735 set to *True*. If the <NewData> element does not exist or is empty, it means than the current data values should be
 1736 removed. The default value for the overrideAllowed attribute is *False*, which means that the <Modification>
 1737 is only allowed to add new data, not to remove or replace existing data. The notChangedSince attribute is used
 1738 to handle concurrent updates. When the notChangedSince attribute is present, the modification is allowed to be
 1739 done only if the data to be modified has not changed since the time specified by the value of the notChangedSince
 1740 attribute. The <Modification> element MUST also have the itemID attribute, when multiple <Modification>
 1741 elements are included in one <Modify> element and partial failure is allowed so that failed parts can be identified.

1742 In addition to the id attribute, the <Modify> element can have also the itemID attribute. This is necessary when
 1743 the request message has multiple <Modify> elements. The response message can refer to itemID attributes of
 1744 the <Modify> elements and so map <ModifyResponse> elements in the response message to the corresponding
 1745 <Modify> elements.

1746 The schema for <Modify>

```

1747
1748
1749 <xs:element name="Modify" type="ModifyType" />
1750 <xs:complexType name="ModifyType">
1751   <xs:sequence>
1752     <xs:group ref="ResourceIDGroup" minOccurs="0" />
1753     <xs:element name="Modification" maxOccurs="unbounded">
1754       <xs:complexType>
1755         <xs:sequence>
1756           <xs:element name="Select" type="SelectType" minOccurs="0" />
1757           <xs:element name="NewData" minOccurs="0">
1758             <xs:complexType>
1759               <xs:sequence>
1760                 <xs:any minOccurs="0" maxOccurs="unbounded" />
1761               </xs:sequence>
1762             </xs:complexType>
1763           </xs:element>
1764         </xs:sequence>
1765         <xs:attribute name="id" type="xs:ID" />
1766         <xs:attribute name="itemID" type="IDType" />
1767         <xs:attribute name="notChangedSince" type="xs:dateTime" />
1768         <xs:attribute name="overrideAllowed" type="xs:boolean" default="0" />
1769       </xs:complexType>
1770     </xs:element>
1771     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1772   </xs:sequence>
    
```

```
1773     <xs:attribute name="id" type="xs:ID"/>
1774     <xs:attribute name="itemID" type="IDType"/>
1775 </xs:complexType>
1776
1777
```

1778 5.2. <ModifyResponse> element

1779 The <ModifyResponse> element contains the <Status> element, which describes whether or not the requested
1780 modification succeeded. There is also a possible time stamp attribute, which provides a time value that can be used
1781 later to check whether there have been any changes since this modification, and an itemIDRef attribute to map the
1782 <ModifyResponse> elements to the <Modify> elements in the request.

1783 The schema for <ModifyResponse>

```
1784
1785
1786 <xs:element name="ModifyResponse" type="ResponseType"/>
1787 <xs:complexType name="ResponseType">
1788   <xs:sequence>
1789     <xs:element ref="Status"/>
1790     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1791   </xs:sequence>
1792   <xs:attribute name="id" type="xs:ID"/>
1793   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
1794   <xs:attribute name="timeStamp" type="xs:dateTime"/>
1795 </xs:complexType>
1796
1797
```

1798 5.3. Processing Rules

1799 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

1800 The <Modify> can contain multiple <Modification> elements. The following rules specify how those must be
1801 supported and handled:

- 1802 • A WSP MUST support one <Modification> element inside a <Modify> and SHOULD support multiple. If
1803 the <Modify> contains multiple <Modification> elements and the WSP supports only one <Modification>
1804 element inside a <Modify>, the processing of the whole <Modify> MUST fail and a status code indicating failure
1805 MUST be returned in the response. The value NoMultipleAllowed SHOULD be used for the second level
1806 status code. If a WSP supports multiple <Modification> element inside a <Modify>, it MAY register the
1807 urn:liberty:dst:multipleModification discovery option keyword.
- 1808 • If the processing of a <Modification> fails even partly due to some reason, depending on the service and/or a
1809 WSP either the processing of the whole <Modify> MUST fail or a WSP MUST try to achieve partial success.
1810 The top level status code Failed or Partial MUST be used to indicate the failure (complete or partial) and a
1811 more detailed second level status code SHOULD be used to indicate the reason for failing to completely process
1812 the failed <Modify> element. Furthermore, the ref attribute of the <Status> element SHOULD carry the value
1813 of the itemID of the failed <Modification> element and in partial success cases it MUST carry this value. The
1814 modifications made based on already processed <Modification> elements of the <Modify> MUST be rolled
1815 back in case of a complete failure. A WSP MUST NOT support multiple <Modification> elements inside one
1816 <Modify>, if it cannot roll back and partial failure is not allowed.
- 1817 • When multiple <Modification> elements inside one <Modify> element are supported and partial success is
1818 allowed, a WSC MUST use the itemID attribute in each <Modification> element so that a WSP can identify
1819 the failed parts, when it is returning status information for a partial success.

1820 What is modified and how depends on a number of parameters including the value of the <Select> element, the
1821 content of the provided <NewData> element, the value of the `overrideAllowed` attribute, and the current content of
1822 the underlying conceptual XML document.

1823 The following rules specify in more detail how modification works:

1824 • When adding new data, the <Select> element will point in the conceptual XML document to an element which
1825 does not exist yet. The new element is added as a result of processing the <Modification> element. In such
1826 cases, when the ancestor elements of the new element do not exist either, they **MUST** be added as part of processing
1827 of the <Modification> element so that processing could be successful.

1828 • If the <Select> points to multiple places and there is a <NewData> element with new values, the processing of
1829 the <Modification> **MUST** fail because it is not clear where to store the new data. If there is no <NewData>
1830 element and the `overrideAllowed` attribute is set to *True*, then the processing of <Modification> can continue
1831 normally, because it is acceptable to delete multiple data elements at once (for example, all `AddressCards`).
1832 When the `overrideAllowed` is set to *False* or is missing, the <NewData> element **MUST** be present as new
1833 data should be added. If the <NewData> element is missing in this case, the processing of the <Modification>
1834 **MUST** fail and the second level status code `MissingNewDataElement` **SHOULD** be returned in addition to top
1835 level status code.

1836 • When there is the <NewData> element with new values and the <Select> points to existing information, the
1837 processing of the <Modification> **MUST** fail, if the `overrideAllowed` attribute is not set to *True*. When
1838 the `overrideAllowed` attribute does not exist or is set to *False*, the new data in the <NewData> element can
1839 only be accepted in two cases: either there is no existing element to which the <Select> points, or there can be
1840 multiple data elements of the same type. This means that, if the <Select> points to an existing container element,
1841 which has a subelement, and only one such container element can exist, the <Modification> **MUST** fail, even
1842 if the only subelement the container element has inside the <NewData> does not yet exist in the conceptual XML
1843 document. The second level status code `ExistsAlready` **SHOULD** be used to indicate in details the reason for
1844 the failure in addition to the top level status code. The lack of those other sub-elements inside the <NewData>
1845 means that they should be removed, which is only possible when `overrideAllowed` attribute equals to *True*.
1846 When there can be multiple elements of the same type, the addition of a new element **MUST** fail, if there exists
1847 already an element of same type have the same value of the distinguishing part. In the case of a personal profile
1848 service, adding a new <AddressCard> element **MUST** fail, if there already exists an <AddressCard> element
1849 which has an `id` attribute of the same value as the provided new <AddressCard> element. The second level status
1850 code `ExistsAlready` **SHOULD** also be used to indicate the detailed reason for failure.

1851 • When all or some of the data inside the <NewData> element is not supported by the WSP, or the provided data is not
1852 valid, the processing of the whole <Modification> **SHOULD** fail and second level status code `InvalidData`
1853 **SHOULD** be returned in the response.

1854 • When the <Modification> element tries to extend the service either by pointing to a new data type behind an
1855 <Extension> element with the <Select> element, or having new sub-elements under an <Extension> element
1856 inside the <NewData> element and the WSP does not support extension in general or for the requesting party, it
1857 **SHOULD** be indicated in the response message with the second level status code `ExtensionNotSupported`.
1858 When the WSP supports extensions, but does not accept the content of the <Select> or <NewData>, then second
1859 level status codes `InvalidSelect` and `InvalidData` **SHOULD** be used as already described.

1860 The common attributes belonging to the attribute groups `commonAttributes` and `leafAttributes` are mainly
1861 supposed to be written by the WSP hosting the data service. There are some additional rules for handling these
1862 common attributes in case of modifications.

- 1863 • When any of the `ACC`, `modifier`, `ACCTime` or `modificationTime` attributes is used in a resource, the WSP
1864 hosting the data service MUST keep their values up to date. When data is modified, the `modifier` MUST contain
1865 the `ProviderID` of the modifier or have no value, and the `modificationTime` MUST define the time of the
1866 modification or have no value. The `ACC` MUST define the attribute collection context of the current value of a data
1867 element or have no value and the `ACCTime` MUST define the time, when the current value of the `ACC` was defined
1868 or have no value.
- 1869 • If the `<NewData>` contains `modifier`, `modificationTime` or `ACCTime` attributes for any data element, the WSP
1870 MUST ignore these and update the values based on other information than those attributes inside the `<NewData>`
1871 provided by the WSC. If the `ACC` attribute is included for any data element, the WSP MAY accept it, depending
1872 on how much it trusts the requesting service provider. The WSP MAY also accept the `id` attribute provided inside
1873 the `<NewData>` and some services MAY require that the `id` attribute MUST be provided by the requesting service
1874 provider.
- 1875 • The `id` attribute MUST NOT be used as a global unique identifier. The value MUST be chosen so that it works
1876 only as unique identifier inside the conceptual XML document, and the value of the `id` attribute SHOULD be kept
1877 the same even if the element is otherwise modified. A WSP MAY not even allow changing the value of the `id`
1878 attribute or any other attribute used to distinguish elements with the same name from each other.
- 1879 • When data is modified based on the `<Modify>` request, the values of the `modificationTime` attributes written
1880 by the WSP hosting the data service MAY be same for all inserted and updated elements, but there is no guarantee
1881 that they will be exactly the same. When the `modificationTime` attribute is used by a data service, the WSP
1882 MUST keep it up to date to indicate the time of the latest modification of an element and update it, when ever
1883 a modification is done either using the `<Modify>` request or some other way. When the `modificationTime`
1884 attribute is used in container elements, the time of a modification MUST be propagated to all ancestor elements of
1885 the modified element all the way up to the root element.
- 1886 Accounting for concurrent updates is handled using the `notChangedSince` attribute inside the `<Modification>`
1887 element.
- 1888 • When the `notChangedSince` attribute is present, the modifications specified by the `<Modification>` element
1889 MUST NOT be made, if any part of the data to be modified has changed since the time specified by the
1890 `notChangedSince` attribute.
1891 The second level status code `ModifiedSince` MUST be used to indicate that the modification was not done
1892 because the data has been modified since the time specified by the `notChangedSince` attribute. If a WSP does
1893 not support processing of this attribute properly, it MUST NOT make any changes and it MUST return the second
1894 level status code `ChangeHistoryNotSupported`. If a WSP supports this `notChangedSince` attribute, it MUST
1895 also support the `changedSince` attribute of the `<QueryItem>` element.
- 1896 A WSC might not be allowed to make certain modifications or any modifications at all.
- 1897 • When a WSP processes the `<Modification>`, it MUST check, whether the resource owner (for example, the
1898 Principal) has given consent to the requester to modify the data. To be able to check WSC-specific access
1899 rights, the WSP MUST authenticate the WSC (see [\[LibertySecMech\]](#) and [\[LibertyMetadata\]](#)). If the consent
1900 check fails for any part of the requested data, the WSP MUST NOT make the modifications requested in the
1901 `<Modification>` element, even when such consent is missing only for some subelement or attribute. The WSP
1902 MAY try to get consent from the Principal while processing the request perhaps using an interaction service (for
1903 more information see [\[LibertyInteract\]](#)). The processing of the `<Modification>` element MUST fail, if the
1904 modification was not allowed. The second level status code `ActionNotAuthorized` MAY be used, if it is
1905 considered that the privacy of the owner of the resource is not compromised. A WSP might check the access rights
1906 at a higher level, before getting to DST processing and MAY return an ID-* Fault Message [\[LibertySOAPBinding\]](#)
1907 and not process the `<Modify>` element at all, if the requesting WSC is not allowed to modify the data.

1908 The WSP may have some restrictions for the data it is hosting.

- 1909
- The schemas for different data services may have some elements for which there is not an exact upper limit on how many can exist. For practical reasons, implementations may set some limits. If a request tries to add more elements than a WSP supports, the WSP will not accept the new element(s) and the processing of the <Modification> element MUST fail. The WSP should use a second level status code NoMoreElements to indicate this specific case.
- 1910
- The schemas for different data services may not specify the length of elements and attributes especially in the case of strings. The WSP may also have limitations of this kind. If a request tries to add longer data elements or attributes than a WSP supports, the WSP may not accept the data and the processing of the <Modification> element will fail. The WSP should use a second level status code DataTooLong to indicate this specific case.
- 1911
- 1912
- 1913
- 1914
- 1915
- 1916
- 1917

1918 5.4. Examples

1919 Please note that as the modification examples are based on the [\[LibertyIDPP\]](#) all DST elements are also in the
1920 namespace defined in [\[LibertyIDPP\]](#).

1921 This example adds a home address to the personal profile of a Principal:

```
1922
1923
1924 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1925   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1926   <pp:Modification>
1927     <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
1928     <pp:NewData>
1929       <pp:AddressCard id='98123'>
1930         <pp:AddressType>urn:liberty:pp:addrType:home</pp:AddressType>
1931         <pp:Address>
1932           <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way North</pp:PostalAddress>
1933           <pp:PostalCode>98503-2341</pp:PostalCode>
1934           <pp:L>Olympia</pp:L>
1935           <pp:ST>wa</pp:ST>
1936           <pp:C>us</pp:C>
1937         </pp:Address>
1938       </pp:AddressCard>
1939     </pp:NewData>
1940   </pp:Modification>
1941 </pp:Modify>
1942
1943
```

1944 The following example replaces the current home address with a new home address in the personal profile of a
1945 Principal. Please note that this request will fail if there are two or more home addresses in the profile, because it
1946 is not clear in this request, which of those addressed should be replaced by this address. In such a case the id attribute
1947 should be used to explicitly point which of the addresses should be changed.

```
1948
1949
1950 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1951   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1952   <pp:Modification overrideAllowed="True">
1953     <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:
1954 home']</pp:Select>
1955     <pp:NewData>
1956       <pp:AddressCard id="98123">
1957         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1958         <pp:Address>
1959           <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way</pp:PostalAddress>
```

```
1960         <pp:PostalCode>98503-2342</pp:PostalCode>
1961         <pp:L>Olympia</pp:L>
1962         <pp:ST>wa</pp:ST>
1963         <pp:C>us</pp:C>
1964     </pp:Address>
1965 </pp:AddressCard>
1966 </pp:NewData>
1967 </pp:Modification>
1968 </pp:Modify>
1969
1970
```

1971 This example replaces the current address identified by an id of '98123' with a new home address, if that address has
1972 not been modified since 12:40:01 21th January 2003 UTC.

```
1973
1974
1975 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
1976     <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
1977     <pp:Modification notChangedSince="2003-01-21T12:40:01Z" overrideAllowed="True">
1978         <pp>Select>/pp:PP/pp:AddressCard[@pp:id='98123']</pp>Select>
1979         <pp:NewData>
1980             <pp:AddressCard id="98123">
1981                 <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
1982                 <pp:Address>
1983                     <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way South</pp:PostalAddress>
1984                     <pp:PostalCode>98503-2398</pp:PostalCode>
1985                     <pp:L>Olympia</pp:L>
1986                     <pp:ST>wa</pp:ST>
1987                     <pp:C>us</pp:C>
1988                 </pp:Address>
1989             </pp:AddressCard>
1990         </pp:NewData>
1991     </pp:Modification>
1992 </pp:Modify>
1993
1994
```

1995 The following example adds another home address to the personal profile of a Principal. An id is provided for the
1996 new address.

```
1997
1998
1999 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
2000     <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
2001     <pp:Modification>
2002         <pp>Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:home']</pp>Select>
2003         <pp:NewData>
2004             <pp:AddressCard id="12398">
2005                 <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
2006                 <pp:Address>
2007                     <pp:PostalAddress>1234 Beach Way</pp:PostalAddress>
2008                     <pp:PostalCode>98765-1234</pp:PostalCode>
2009                     <pp:L>Olympia</pp:L>
2010                     <pp:ST>wa</pp:ST>
2011                     <pp:C>us</pp:C>
2012                 </pp:Address>
2013             </pp:AddressCard>
2014         </pp:NewData>
2015     </pp:Modification>
2016 </pp:Modify>
2017
2018
2019
```

2020 The following example removes all current home addresses from the personal profile of a Principal:


```
2021
2022
2023 <pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
2024   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
2025   <pp:Modification overrideAllowed="True">
2026     <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:
2027 home']</pp:Select>
2028   </pp:Modification>
2029 </pp:Modify>
2030
2031
```

2032 The response for a valid <Modify> is as follows:

```
2033
2034
2035 <pp:ModifyResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
2036 timeStamp="2003-03-23T03:40:00Z">
2037   <pp:StatusCode="OK"/>
2038 </pp:ModifyResponse>
2039
2040
```

2041 **6. Subscriptions and Notifications**

2042 The subscriptions are a mechanism through which WSCs can request for notifications when specified event happens.
2043 The basic case is subscribing to change notifications to get updates when the data hosted by a data service related to a
2044 Principal changes. A WSC may subscribe to change notifications even before the data exists. E.g. a WSC may want to
2045 know, when a Principal adds an email address to her profile, and get that email address to be able to send email to the
2046 Principal. The change of data is not the only possible reason for a notification, there can be service specific triggers
2047 for notifications, e.g. periodic notifications containing current values and notifications after a Principal switches on
2048 her terminal.

2049 As the notifications do not only reveal the data they are carrying, but also that certain thing has just happened, WSPs
2050 must be even more careful to make sure they honor the privacy of the Principals.

2051 Subscriptions and notifications have two basic elements: <Subscription> and <Notification>. These basic
2052 elements used by subscription and notification protocols are introduced first. After that the protocols for subscriptions
2053 and notifications are defined.

2054 **6.1. Basic elements**

2055 The basic elements <Subscription> and <Notification> are transferred using different protocol messages and
2056 can have slightly different content depending on a message, e.g. all the parameters of the <Subscription> element
2057 are not needed in each message. The basic elements are defined here, but their use is defined together with protocol
2058 elements.

2059 **6.1.1. <Subscription> element**

2060 The <Subscription> element contains all the parameters for a subscription. It defines, what data a WSC wants to
2061 have, where it should be sent, when a subscription expires, which events should trigger notifications, etc.

2062 The first parameter inside the <Subscription> element is the <Select> element. This is the same element used
2063 also when querying and modifying data. It defines, what data a notification should return. The use of the <Select>
2064 element inside the <Subscription> element might be a bit different than its use when querying and modifying. The
2065 specifications for services MUST specify possible differences.

2066 The same <ChangeFormat> element, which is used for querying for changes, can be used, when subscribing to
2067 change notifications. This element is used to indicate, which change formats a WSC supports and which it prefers.

2068 The element <NotifyTo> contains the information a WSP needs to be able to send the requested notifications to
2069 a requesting WSC. The type definition for the <NotifyTo> element is imported from [\[LibertySOAPBinding\]](#) and
2070 the element contains three different types of subelements. The <sb-ext:Endpoint> element contains the address
2071 to which notifications MUST be sent. There can also be one or more <sb-ext:SecurityMechID> elements
2072 indicating which security mechanisms a WSP may use (it MUST NOT use any other). A WSC may also provide
2073 credentials, which a WSP MUST use, when sending notifications. The credentials are provided using one or more
2074 <sb-ext:Credential> elements. Note that the type definition imported from [\[LibertySOAPBinding\]](#) also contains
2075 attributes. Attributes *S.actor* and *S.mustunderstand* MUST NOT be used in a <NotifyTo> element. A WSC
2076 can provide a WSP different information for sending normal notifications and for sending notifications telling that the
2077 subscription has ended. The information for sending end notifications is provided in a <NotifyEndedTo> element,
2078 which is optional and has same type definition as the <NotifyTo> element. Please note that attributes *S.actor*
2079 and *S.mustunderstand* MUST NOT be used in a <NotifyEndedTo> element either. If the <NotifyEndedTo>
2080 element is not used, the same information provided for normal notifications is also used for end notifications. The
2081 purpose of the <NotifyEndedTo> element is to make it possible to receive notifications in one point and manage
2082 subscriptions in another point, when end notifications are used.

2083 There can be different type of notifications. E.g. a notification can be sent immediately or multiple notification could
2084 be sent in a bigger batch. The element <Type> defines, what type of notifications a WSC is requesting. The element

2085 <Type> element is of type element `TypeType` and this type MUST be specified by services including the detailed
2086 semantics and allowed values.

2087 The normal reason for a notification is that the data addressed by the <Select> element has changed. There can be
2088 also other reasons triggering notifications. The <Trigger> element contains those triggers. The <Trigger> element
2089 is of type element `TriggerType`. This type MUST be defined in the services schemas and the service specifications
2090 MUST define semantics and values for this parameter. When the <Trigger> element is not used, a WSC is requesting
2091 normal change notifications unless otherwise specified by a service specification.

2092 A subscription is not valid forever. The `starts` attribute defines the time after which a subscription is valid and
2093 notifications can be sent, if the triggering event occurs. The `starts` attribute MUST be used only, when a subscription
2094 is not supposed to be valid immediately after processing the request. The `expires` attribute defines the time, when a
2095 subscription expires, if not renewed before that time. Instead of the `expires` attribute the `duration` attribute may
2096 be used to indicate the duration of a subscription.

2097 The different subscriptions related to a same resource are distinguished from each other by IDs. A WSC uses
2098 the `invokeID` attribute, when creating a new subscription. After creating a subscription, it is referred by the
2099 `subscriptionID` attribute. A WSP gives value to this attribute.

2100 Usually a notification contains data related to a resource. Sometimes a notification could be used to indicate that
2101 an event related to a resource has happened, e.g. the data addressed by the <Select> element has changed, without
2102 containing the changed data. The attribute `includeData` defines, should the data be included or not. The default value
2103 is `Yes`. Other possible values are: `No` (no data is returned) and `YesWithCommonAttributes` (the data is returned
2104 with the common attributes).

2105 The use of the different parameters of a subscription are defined in more detail with the protocol elements and
2106 processing rules related to those. The schema for the <Subscription> element is as follows:

```
2107
2108 <xs:element name="Subscription">
2109   <xs:complexType>
2110     <xs:sequence>
2111       <xs:element name="Select" type="SelectType" minOccurs="0"/>
2112       <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
2113       <xs:element name="NotifyTo" type="xs:anyURI"/>
2114       <xs:element name="NotifyEndedTo" type="xs:anyURI" minOccurs="0"/>
2115       <xs:element name="Type" type="TypeType" minOccurs="0"/>
2116       <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
2117       <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2118     </xs:sequence>
2119     <xs:attribute name="starts" type="xs:dateTime"/>
2120     <xs:attribute name="expires" type="xs:dateTime"/>
2121     <xs:attribute name="duration" type="xs:duration"/>
2122     <xs:attribute name="id" type="xs:ID"/>
2123     <xs:attribute name="invokeID" type="IDType"/>
2124     <xs:attribute name="subscriptionID" type="IDType"/>
2125     <xs:attribute name="includeData" default="Yes">
2126       <xs:simpleType>
2127         <xs:restriction base="xs:string">
2128           <xs:enumeration value="Yes"/>
2129           <xs:enumeration value="No"/>
2130           <xs:enumeration value="YesWithCommonAttributes"/>
2131         </xs:restriction>
2132       </xs:simpleType>
2133     </xs:attribute>
2134   </xs:complexType>
2135 </xs:element>
2136
```

2137 6.1.2. <Notification> element

2138 The <Notification> element is the basic element, when either a normal notification or notification to indicate the
2139 end of a subscription is sent. The main content of the <Notification> element is the <Data> element, which
2140 contains the data the notification carries, e.g. the current location or the changed home address. In a case of a change
2141 notification same formats as in responses to queries for changed data are used.

2142 The <Data> element may also contain some other type of data indicating what kind of an event has happened. The
2143 whole <Data> element might not be used at all as it is possible to subscribe to notifications to indicate that an event
2144 has happened, e.g. data has changed without having the data in the notification message.

2145 In addition to the <Data> element the <Notification> element has a number of attributes. There are two ID
2146 attributes: invokeID and subscriptionID. The invokeID attribute is only used to in the initial reply to the
2147 subscription and after that the subscriptionID attribute is used.

2148 The changeFormat attribute MUST be used to indicate the used format for data changes in case of a change
2149 notification unless only one format is specified to be used for the service type in question.

2150 The expires attribute is used to indicate in a notification message the time, when the subscription will expire.
2151 Instead of the expires attribute the duration attribute may be used to indicate, how long the subscription is
2152 still valid. In an end notification the endReason attribute can be used to indicate the reason for the end of the
2153 subscription. This might give some indication to a WSC that a WSP is having some problems or whether it makes
2154 sense or not for a WSC to try to make the subscription again. The endReason attribute is not used in normal
2155 notifications, only when notifying that a subscription has ended. Possible values for the endReason attribute include:
2156 urn:liberty:dst:endreason:unspecified, urn:liberty:dst:endreason:wscnotacknowledging,
2157 urn:liberty:dst:endreason:resourcedeleted, urn:liberty:dst:endreason:expired and
2158 urn:liberty:dst:endreason:credentialexpired. A WSP must be careful not to compromise the
2159 privacy of a Principal, when sending the reason codes for ending a subscription.

2160 The schema for the <Notification> element is as follows:

```
2161 <xs:element name="Notification">
2162   <xs:complexType>
2163     <xs:sequence>
2164       <xs:element name="Data" minOccurs="0">
2165         <xs:complexType>
2166           <xs:sequence>
2167             <xs:any minOccurs="0" maxOccurs="unbounded" />
2168           </xs:sequence>
2169         </xs:complexType>
2170       </xs:element>
2171     </xs:sequence>
2172     <xs:attribute name="id" type="xs:ID" />
2173     <xs:attribute name="invokeID" type="IDType" />
2174     <xs:attribute name="subscriptionID" type="IDType" />
2175     <xs:attribute ref="changeFormat" />
2176     <xs:attribute name="expires" type="xs:dateTime" />
2177     <xs:attribute name="duration" type="xs:duration" />
2178     <xs:attribute name="endReason" type="xs:anyURI" />
2179   </xs:complexType>
2180 </xs:element>
2181
2182
```

2183 6.2. Protocol elements

2184 There are eight different protocol elements defined in following subchapters, half of them are responses to
2185 the others. A WSC may send the <Subscribe> and <QuerySubscription> elements and receive back the
2186 <SubscribeResponse> and <Subscriptions> elements. Based on existing subscriptions a WSP may sent
2187 to a WSC <Notify> and <Ended> elements and may get back as acknowledgements <NotifyResponse> and
2188 <EndedResponse> elements.

2189 6.2.1. Subscribing notifications and modifying subscriptions

2190 6.2.1.1. <Subscribe> element

2191 The <Subscribe> element is used to subscribe to notifications, modify existing subscriptions, renew subscriptions
2192 which are about to expire and cancel subscriptions which a WSC does not need any more.

2193 The <Subscribe> element has two main parts, first it has either the <ResourceID> element or the
2194 <EncryptedResourceID> element to specify the resource in question and then the <Subscription> ele-
2195 ments. One <Subscribe> element can have multiple <Subscription> elements related to the same resource.
2196 The <Subscribe> element has also the id and itemID attributes to be used in the same way as they are used in
2197 other request messages. Also one new attribute is defined: returnCurrentValues, the default value is True. If a
2198 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
2199 False. Please note that there are cases in which this attribute is meaningless. If the start time has been defined for a
2200 subscription, no current values are returned by the response.

2201 The content of the <Subscription> elements depends, what a WSC wants to do. The rules are simple:

- 2202 • For new subscription, out of invokeID and subscriptionID attributes there is only the invokeID attribute.
2203 The subscriptionID attribute MUST NOT be used, when a WSC requests for a new subscription. The rest of
2204 parameters are used normally.
- 2205 • For renewing an existing subscription the subscriptionID attribute MUST be used to identify the right
2206 subscription and the expires or the duration attribute MUST be used to indicate the requested lifetime of
2207 the subscription. The id MAY be used, if needed. As renewing can be combined with other modifications to an
2208 existing subscription, other content is also allowed except the invokeID attribute, which MUST NOT be used.
- 2209 • For canceling an existing subscription, a WSC MUST use the subscriptionID attribute to identify the
2210 subscription and no other content is allowed, except the id MAY be used, if needed.
- 2211 • For modifying an existing subscription, the subscriptionID attribute MUST be used to identify the right
2212 subscription and the changed information MUST be provided. If any of the elements, <Select>, <Type>,
2213 <Trigger> or <Extension> has changed, the element must be provided with the requested current content.
2214 This means that if one of the triggers has changed and two other triggers haven't, they all must be provided as
2215 part of the <Trigger> element. In case some of those four elements doesn't have any content anymore, an empty
2216 element must be provided to indicate the deletion, e.g. <Trigger/>. If there is no changes e.g. to <Trigger>,
2217 the <Trigger> element is not present in the message.

2218 The schema for the <Subscribe> element is as follows:

```
2219 <xs:element name="Subscribe" type="SubscribeType"/>
2220 <xs:complexType name="SubscribeType">
2221   <xs:sequence>
2222     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2223     <xs:element ref="Subscription" maxOccurs="unbounded"/>
2224     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2225   </xs:sequence>
2226   <xs:attribute name="id" type="xs:ID"/>
2227   <xs:attribute name="itemID" type="IDType"/>
2228   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
2229 </xs:complexType>
2230
2231
```

2232 6.2.1.2. <SubscribeResponse> element

2233 The response to the <Subscribe> has two main parts. There are the <Status> element and the <Notification>
2234 element(s). The <Notification> element is allowed to have the invokeID attribute only, when it is the response

2235 to the initial subscription, after that there is no `invokeID` associated with the subscription as the `subscriptionID`
2236 is used. So, when a WSP is responding to subscription modifications, renewal and deletion, there **MUST NOT** be an
2237 `invokeID` attribute. The `timeStamp` attribute is provided to be used e.g. for the value of the `changedSince` attribute
2238 in `<Query>` after the subscription has expired.

2239 The schema for the `<SubscribeResponse>` element is as follows:

```
2240  
2241 <xs:element name="SubscribeResponse" type="SubscribeResponseType" />  
2242 <xs:complexType name="SubscribeResponseType">  
2243   <xs:sequence>  
2244     <xs:element ref="Status" />  
2245     <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />  
2246     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />  
2247   </xs:sequence>  
2248   <xs:attribute name="id" type="xs:ID" />  
2249   <xs:attribute name="itemIDRef" type="IDReferenceType" />  
2250   <xs:attribute name="timeStamp" type="xs:dateTime" />  
2251 </xs:complexType>  
2252
```

2253 6.2.1.3. Processing rules

2254 NOTE: The common processing rules specified earlier **MUST** also be followed (see [Section 3](#)).

2255 The same `<Subscribe>` element can be used for a number of tasks: to create a new subscription, to renew, modify
2256 and cancel an existing subscription.

2257 • When a WSC has included the `invokeID` attribute in a `<Subscription>` element, this `<Subscription>`
2258 element is used to create a new subscription. A WSC **MUST NOT** put a `subscriptionID` attribute into the
2259 same `<Subscription>` element. If both the `subscriptionID` and the `invokeID` attributes are in the same
2260 `<Subscription>` element sent by a WSC, the receiving WSP **MUST** discard that `<Subscription>` element
2261 and **SHOULD** use in the response the second level status code `NewOrExisting` to indicate that it has failed to
2262 process the `<Subscription>` element as it has not been clear does a WSC want to create a new subscription or
2263 modify an existing subscription.

2264 • When a new subscription is created, a WSP **MUST** return the `invokeID` attribute with the same value as the
2265 requesting WSC used for it. A WSP **MUST** also create a new ID for the subscription to be used in the notifications
2266 and further requests. This ID **MUST** be returned in the `subscriptionID` attribute of the `<Notification>`
2267 element return in the `<SubscribeResponse>` related to the `<Subscribe>` element in the request message.

2268 • When a WSC wants to cancel an existing subscription, the `<Subscription>` element **MUST** have only the
2269 `subscriptionID` attribute to indicate, which subscriptions a WSC wants to cancel. An `ID` attribute **MAY**
2270 also be used, if needed. If a WSC wants to cancel all subscriptions related to the same resource, it **MUST**
2271 use an empty `<Subscription>` element (i.e. `<Subscription/>` element) which **MUST NOT** have the
2272 `subscriptionID` attribute. Again an `ID` attribute **MAY** also be used, if needed. If any other content in addition
2273 to the `subscriptionID` attribute and possible `id` attribute is used, a WSP **MUST** process the `<Subscription>`
2274 element as it would be a request to modify or renew an existing subscriptions.

2275 • When a WSC wants to modify an existing subscription, it **MUST** send a `<Subscription>` element which has
2276 the `subscriptionID` referring to an existing subscription. Only the changed values **MUST** be included in the
2277 `<Subscription>` element. E.g. if a WSC does not want to change the type of the subscription, it does not
2278 need to include the `<Type>` element. The existing values of a subscription for the parameters included in the
2279 `<Subscription>` element **MUST** be overwritten with the values provided in the `<Subscription>` element, if
2280 a WSP supports the provided new values (see rest of the processing rules in this chapter and also the common
2281 processing rules in [Section 3](#)).

- 2282 For each successful <Subscription> element in the request a WSP MUST return a <Notification> element in
2283 the response, even when no data is returned to confirm the expiration time/duration of the subscription.
- 2284 • A <Notification> element MUST have the subscriptionID of the subscription it is related to. If the
2285 <Subscription> element was sent to cancel all existing subscription, the corresponding <Notification>
2286 element in the <SubscribeResponse> MUST NOT have the subscriptionID attribute at all.
- 2287 • The attribute endReason MUST NOT be used in a <Notification> element, when sending it in a
2288 <SubscribeResponse> element. If a subscription was not accepted, the reason is indicated with status codes.
- 2289 A subscription must contain the address to which notifications should be sent and other information needed by a WSP
2290 to be able to send the requested notifications.
- 2291 • If the <NotifyTo> element is missing from a <Subscription> element, the processing of the <Subscription>
2292 MUST fail and a second level status code MissingNotifyToElement indicating this SHOULD be used in
2293 addition to the top level status code.
- 2294 • The <NotifyTo> element MUST contain at least the <Endpoint> element, which MUST contain a complete
2295 URI. If the <Endpoint> element is missing from a <NotifyTo> element, the processing of the <Subscription>
2296 containing this <NotifyTo> element MUST fail and a second level status code MissingEndpointElement
2297 indicating this SHOULD be used in addition to the top level status code. Also, if a WSP is able to verify the validity
2298 of the address provided in the <Endpoint> element and finds out that the address is not valid, the processing of
2299 the <Subscription> MUST fail and a second level status code InvalidEndpoint indicating this SHOULD be
2300 used in addition to the top level status code.
- 2301 • A WSC SHOULD also provide one or more <SecurityMechID> elements to indicate, which security mech-
2302 anisms it supports. The <SecurityMechID> elements MUST be specified according to [\[LibertySecMech\]](#)
2303 and the mechanisms SHOULD be listed in the order of preference by the WSC. If a WSC does not provide
2304 <SecurityMechID> element, the default is urn:liberty:security:2003-08:null:null, which MUST be
2305 avoided, when releasing privacy sensitive data (the case almost every time with identity services). A WSP MUST
2306 refuse to use mechanisms, which it considers to be not good enough, also Principal owning the resource may have
2307 set some security requirements, which a WSP MUST follow. If a WSP wants to get the <SecurityMechID>
2308 element, but it is not included, the processing of the <Subscription> element MUST fail and a second level
2309 status code MissingSecurityMechIDElement indicating this SHOULD be used in addition to the top level
2310 status code. If a WSP does not support or accept any of the proposed security mechanisms, it MUST NOT accept
2311 the subscription and so the processing of the <Subscription> element MUST fail and a second level status
2312 code SecurityMechIDNotAccepted indicating this SHOULD be used in addition to the top level status code.
2313 Some of the security mechanisms require also credentials and a WSC MUST provide those also. If credentials are
2314 needed, but they are not provided, the processing of the <Subscription> element MUST fail and a second level
2315 status code MissingCredentials indicating this SHOULD be used in addition to the top level status code.
- 2316 • The information provided in a <NotifyTo> element is also used, when sending end notifications unless separate
2317 <NotifyEndedTo> element is provided. The information in a provided <NotifyEndedTo> element replaces
2318 the information in a <NotifyTo> element. Please note that a <NotifyEndedTo> element might contain only a
2319 different endpoint for end notifications, in which case the same security mechanisms and credentials provided in
2320 a <NotifyTo> element are used. The processing rules for the subelements of a <NotifyEndedTo> element are
2321 the same as for subelements of a <NotifyTo> element except the <Endpoint> element is not mandatory as the
2322 value provided in a <NotifyTo> element may be used (see previous paragraph).
- 2323 • Please note that if needed credentials expire earlier than a subscription is suppose to expire and a WSC does
2324 not provide new credentials before they expire, the subscription will expire as a WSP is not able to send the
2325 notifications anymore.

- 2326 There might be different type of notifications and different triggers causing those. A WSP may not support all different
2327 type of features available.
- 2328 • A WSP MUST follow the processing rules defined in the specification for the service a WSP hosting for the
2329 elements <Type> and <Trigger>. If the use of these elements is not specified for the service, but either
2330 of both of them are included in a <Subscription> element in a <Subscribe> request, the processing of
2331 the <Subscription> MUST fail and a second level status code indicating this SHOULD be used, either
2332 `TypeNotSupported` or `TriggerNotSupported`.
- 2333 • If a WSP does not support the <Type> of the notification a WSC requests, the processing of the <Subscription>
2334 MUST fail and a second level status code `TypeNotSupported` indicating this SHOULD be used in addition to the
2335 top level status code. Similarly if a WSP does not support the <Trigger> a WSC requests, the processing of the
2336 <Subscription> MUST fail and a second level status code `TriggerNotSupported` indicating this SHOULD
2337 be used in addition to the top level status code.
- 2338 A WSC may request, when the first notification may be sent and when a subscription should expire.
- 2339 • If a <Subscription> element contains a `starts` attribute, that subscription, if accepted, MUST be valid after
2340 the time defined by this `starts` attribute. If there is no `starts` attribute used, then that subscription, if accepted
2341 by a WSP, MUST be valid immediately after processing the request. Also, if the time specified by the `starts`
2342 attribute is in the past, then that subscription, if accepted by a WSP, MUST be valid immediately after processing
2343 the request.
- 2344 • The time specified by the `expires` attribute MUST be the same time or a later time than the time specified by the
2345 `starts` attribute in the same <Subscription> element. It also MUST be later than the current time. If either of
2346 the checks is not passed, then the processing of the <Subscription> MUST fail and a second level status code
2347 `InvalidExpires` indicating this SHOULD be used in addition to the top level status code.
- 2348 • A WSP MAY change the time when a subscription expires from the expiration time requested by a WSC with
2349 the `expires` or `duration` attribute. A WSP MAY shorten the expiration time, but it MUST NOT make the
2350 expiration time longer. If no `expires` or `duration` attribute is not included in a <Subscription> element in a
2351 <Subscribe> request from a WSC, a WSP MUST specify the expiration time for the subscription, if expiration
2352 times are required either by the service specification or the WSP. A WSP MUST use the same attribute a WSC
2353 used in a request to specify the expiration time (`expires` or `duration`), unless otherwise specified in a service
2354 specification. A service specification MAY define that only either of those may be used.
- 2355 • If a WSC wants to renew an existing subscription, which is about to end, it MUST modify that subscription and
2356 give a new value for the `expires` or `duration` attribute of that subscription. A WSP MAY modify the new value
2357 in the same way as it MAY modify the proposed value for a new subscription.
- 2358 • When expiration times are used, a WSP MUST include the `expires` or `duration` attribute with the current value
2359 in each <Notification> element included in a <SubscribeResponse> element.
- 2360 • There are two special cases available using subscriptions expirations. When the `starts` and the `expires`
2361 attributes have exactly same values, the meaning is that a notification MUST be sent exactly at that time. When the
2362 `duration` attribute is set to zero (e.g. "P0D"), the subscription expires immediately after first notification. Note:
2363 when a WSC wants to create subscription, which is valid only for a very short time it MUST explicitly specify a
2364 very short duration and not try to use a value equally to zero.
- 2365 A WSC may want to get the current values a the data when subscribing to notifications.

- 2366 • The default value of the `returnCurrentValues` attribute is `Yes`, so by default a WSP MUST include the current
2367 values in the response. If a WSC sets the value of the `returnCurrentValues` to `No`, a WSP MUST NOT return
2368 the current values in the response. Also if the `includeData` in the `<Subscription>` element is set to `No`, the
2369 data for that subscription MUST NOT be included in the response even, if the `returnCurrentValues` attribute
2370 has the value `Yes`.
- 2371 • The current values of the data addressed by the `<Select>` element of a `<Subscription>` element are included in
2372 the `<Data>` element of the corresponding `<Notification>` element. The data inside the `<Data>` element MUST
2373 be selected in the same way as in a normal query (see [Section 4](#)). If the `includeData` in the `<Subscription>`
2374 has the value `YesWithCommonAttributes`, the existing common attributes MUST be returned with the data.
- 2375 • A WSP MUST NOT return the `<Data>` element, when a WSC is modifying, renewing or canceling existing
2376 subscription unless a WSC is modifying the `<Select>` element of a subscription.
- 2377 The access and privacy policies specified by the resource owner may not allow a WSC to subscribe to the data of a
2378 resource or to some events related to a resource.
- 2379 • When a WSP processes a `<Subscription>` element, it MUST check whether the resource owner (the Principal,
2380 for example) has given consent to return the requested information in notification messages. To be able to check
2381 WSC-specific access rights, the WSP MUST authenticate the WSC (see [\[LibertySecMech\]](#) and [\[LibertyMeta-](#)
2382 [data\]](#)). The WSP MUST also check that any usage directive given in the request is acceptable based on the usage
2383 directives defined by the resource owner (see [\[LibertySOAPBinding\]](#)). If either check fails, the WSP MUST NOT
2384 accept the subscription and the processing of that `<Subscription>` MUST fail. The WSP MAY try to get consent
2385 from the Principal while processing the request, perhaps by using an interaction service (see [\[LibertyInteract\]](#)). A
2386 WSP might check the access rights and policies in usage directives at a higher level, before getting to DST pro-
2387 cessing and MAY, in this case, just return an ID-* Fault Message [\[LibertySOAPBinding\]](#) without processing the
2388 `<Subscribe>` element at all, if the requesting WSC is not allowed to subscribe to data or event in question.
- 2389 • Note that there can be consent for subscribing to some data element, but not its attributes. A Principal might not
2390 want to release the `modifier` attribute, if she does not want to reveal information about which services she uses.
2391 If a WSC is not allowed to get all the data, but some data it wants, a WSP SHOULD accept the subscription, but it
2392 MAY also reject it. If a subscription is accepted, the data for which there is no consent from the Principal MUST
2393 be handled as if there was no data, i.e. that data MUST NOT be returned in the response message, even if the
2394 current values should be returned. Also that data MUST NOT be included in the notification messages sent later
2395 on.
- 2396 • If a WSC has made a subscription and included the usage directive it has promised to obey and later wants to
2397 change the usage directive, it MUST cancel the subscription and make a new subscription with the new value for
2398 the usage directive.

2399 6.2.2. Querying existing subscriptions

2400 6.2.2.1. `<QuerySubscriptions>` element

2401 The existing subscriptions can also be queried. The purpose is to list all the currently active subscriptions the requesting
2402 WSC has related to the specific resource. Either the `<ResourceID>` element or the `<EncryptedResourceID>`
2403 element is used to identify the resource.

2404 The schema for the `<QuerySubscriptions>` element is as follows:

```
2405 <xs:element name="QuerySubscriptions" type="QuerySubscriptionsType" />
2406 <xs:complexType name="QuerySubscriptionsType">
2407   <xs:sequence>
2408     <xs:group ref="ResourceIDGroup" minOccurs="0" />
```

```
2410         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2411     </xs:sequence>
2412     <xs:attribute name="id" type="xs:ID" />
2413     <xs:attribute name="itemID" type="IDType" />
2414 </xs:complexType>
2415
```

2416 **6.2.2.2. <Subscriptions> element**

2417 The response to the <QuerySubscriptions> is a list of <Subscription> elements inside a <Subscriptions>
2418 element. The invokeID MUST NOT be used in <Subscription> elements returned.

2419 The <Status> element is also included in the <Subscriptions> element to indicate, how the processing of the
2420 <QuerySubscriptions> succeeded.

2421 The schema for the <Subscriptions> element is as follows:

```
2422
2423 <xs:element name="Subscriptions" type="SubscriptionsType" />
2424 <xs:complexType name="SubscriptionsType">
2425     <xs:sequence>
2426         <xs:element ref="Status" />
2427         <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded" />
2428         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2429     </xs:sequence>
2430     <xs:attribute name="id" type="xs:ID" />
2431     <xs:attribute name="itemIDRef" type="IDReferenceType" />
2432 </xs:complexType>
2433
```

2434 **6.2.2.3. Processing rules**

2435 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

- 2436 • A WSP MUST return in a response to a <QuerySubscriptions> all valid subscriptions a WSC has related to
2437 the specified resource at the WSP. One <Subscription> element MUST be used for each valid subscription the
2438 request WSC has.
- 2439 • If a WSC does not have any valid subscriptions at a WSP related to the specified resource, the WSP MUST NOT
2440 return any <Subscription> element in the <Subscriptions> element.
- 2441 • A <Subscription> element in a <Subscriptions> element MUST have all the parameters currently valid for
2442 it.
- 2443 • A <Subscription> element in a <Subscriptions> element MUST have a subscriptionID attribute and
2444 MUST NOT have a invokeID attribute as a invokeID is not valid after the response to the original request to
2445 create a new subscription.

2446 6.2.3. Sending notifications

2447 6.2.3.1. <Notify> element

2448 The subscriptions are made to get the notifications about data changes and other events. The <Notify> element
2449 is used to carry the <Notification> elements. A <Notification> element inside a <Notify> element is not
2450 allowed to have the endReason attribute as the subscription has not ended. Ending must be indicated with the Ended
2451 message. The timeStamp attribute is provided to be used e.g. for the value of the notChangedSince attribute in
2452 <Modify> after the subscription has expired.

2453 The schema for the <Notify> element is as follows:

```
2454  
2455 <xs:element name="Notify" type="NotifyEndedType" />  
2456 <xs:complexType name="NotifyEndedType">  
2457 <xs:sequence>  
2458 <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />  
2459 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />  
2460 </xs:sequence>  
2461 <xs:attribute name="id" type="xs:ID" />  
2462 <xs:attribute name="itemID" type="IDType" />  
2463 <xs:attribute name="timeStamp" type="xs:dateTime" />  
2464 </xs:complexType>  
2465
```

2466 6.2.3.2. <NotifyResponse> element

2467 The response to a <Notify> acknowledges the received <Notification> elements and so it just contains the
2468 <Status> element.

2469 The <Notify> messages may not always be acknowledged. A service specification MUST define are these
2470 acknowledgements used or not or is it an implementation/deployment specific decision.

2471 The schema for the <NotifyResponse> element is as follows:

```
2472  
2473 <xs:element name="NotifyResponse" type="NotifyEndedResponseType" />  
2474 <xs:complexType name="NotifyEndedResponseType">  
2475 <xs:sequence>  
2476 <xs:element ref="Status" />  
2477 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />  
2478 </xs:sequence>  
2479 <xs:attribute name="id" type="xs:ID" />  
2480 <xs:attribute name="itemIDRef" type="IDReferenceType" />  
2481 </xs:complexType>  
2482
```

2483 6.2.3.3. Processing rules

2484 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

- 2485 • A WSP MUST send a notification message to a WSC, which has made a subscription, when an event defined by
2486 the parameters of that subscription happens.
- 2487 • When sending a notification message to a WSC, a WSP MUST use the information provided in the <NotifyTo>
2488 element (endpoint, security mechanism and credentials).

- 2489 • If the receiving WSC can not process successfully one of the <Notification> elements inside one <Notify>
2490 element, it SHOULD process normally the rest of the <Notification> elements and try to achieve a partial
2491 success. A WSC MUST support multiple <Notification> elements inside one <Notify> element.
- 2492 • A <Notification> element inside a notification message MUST NOT have an invokeID or a endReason
2493 attribute. If there are a WSC MUST discard them.
- 2494 • A <Notification> element inside a notification message MUST have a subscriptionID attribute to iden-
2495 tify the subscription based on which the notification message is sent. If the subscriptionID attribute
2496 is missing, the processing of that <Notification> element MUST fail and a second level status code
2497 MissingSubscriptionID SHOULD be used in addition to a top level status code. If a WSC does not rec-
2498 ognize the value of a subscriptionID attribute, the processing of that <Notification> element MUST fail
2499 and a second level status code InvalidSubscriptionID SHOULD be used in addition to a top level status code.
- 2500 • A <Notification> element inside a notification message MUST have the expires or the duration attribute,
2501 when subscription expiration is used. When a WSC receiving a notification knows that the expires or
2502 the duration attribute should have been used, but it is not, it SHOULD use the second level status code
2503 MissingExpiration. A WSC MUST decide is this a failure or not, but it SHOULD anyway indicate to a
2504 WSP that it was expecting the expires or the duration attribute.
- 2505 • If a <Notification> element is supposed to contain data about a resource (i.e. the includeData attribute of a
2506 subscription has either the value Yes or YesWithCommonAttributes), the <Data> element MUST be used in a
2507 <Notification> element. The content of a <Data> element MUST be according to the parameters of the related
2508 subscription and the related event, which has caused this <Notification> element to be sent inside a notification
2509 message. In case of a change notification the same formatting rules for the content as in case of a query for changes
2510 MUST be followed (see Section 4). A WSP MUST NOT include any data, which the WSC is not allowed to get
2511 based on access rights and privacy policies defined by the resource owner. If a <Data> element should have
2512 been included in a <Notification> element, but it is missing, the processing of the <Notification> element
2513 MUST fail and a second level status code MissingDataElement SHOULD be used in addition to the top level
2514 status code.
- 2515 • For change notification a changeFormat attribute MUST be added for a <Data> element to indicate the format
2516 used to shown the changes, if a service specification has not mandated only one specific format to be used for this.
- 2517 • If the data inside a <Data> element is invalid, the processing of the <Notification> element MUST fail and a
2518 second level status code InvalidData SHOULD be used in addition to the top level status code. A WSC MUST
2519 accept all the data, which can be consider as possible normal extension, if extensions are allowed for a service
2520 based on the service specification.
- 2521 • A WSP SHOULD resend a notification for which it does not get an acknowledgement in reasonable time, if
2522 acknowledgements are used. If a WSP does not get acknowledgments at all within its time and other limits, it
2523 MAY cancel the related subscription.

2524 6.2.4. Notifying the end of a subscription

2525 6.2.4.1. <Ended> element

2526 There is also another kind of a notification message. The <Ended> message indicates that the subscription has ended
2527 for one reason or another. The content of the <Ended> is the same as for the <Notify> with the difference that now
2528 the endReason attribute must be present in the <Notification> element(s) and no <Data> element is used.

2529 The schema for the <Ended> element is as follows:

```
2530 <xs:element name="Ended" type="NotifyEndedType" />  
2531  
2532
```

2533 **6.2.4.2. <EndedResponse> element**

2534 The response to the <Ended> is similar to the response to the <Notify>, so mainly only the <Status> element.

2535 The schema for the <EndedResponse> element is as follows:

```
2536  
2537 <xs:element name="EndedResponse" type="NotifyEndedResponseType"/>  
2538
```

2539 **6.2.4.3. Processing rules**

2540 NOTE: The common processing rules specified earlier MUST also be followed (see [Section 3](#)).

- 2541 • When a subscription is not anymore valid, a WSP MUST send an end notification to indicate this, if end
2542 notifications are used. The information provided in the <NotifyEndedTo> element of the subscription MUST be
2543 used. If the <NotifyEndedTo> element has not been used or it doesn't contain all the information, the information
2544 provided in the <NotifyTo> element of the subscription is used to complement.
- 2545 • A <Notification> element inside an end notification message MUST have the attributes endReason and
2546 subscriptionID and it MAY also have the id attribute, if needed, but it MUST NOT have any other content. If
2547 it has, the receiving WSC SHOULD ignore the other content.
- 2548 • The value of an endReason attribute SHOULD be either urn:liberty:dst:endreason:unspecified,
2549 urn:liberty:dst:endreason:expired, urn:liberty:dst:endreason:credentialsexpired,
2550 urn:liberty:dst:endreason:wscnotacknowledging or urn:liberty:dst:endreason:resourcedeleted.
2551 It MAY also have some service or implementation specific value. A WSP MUST be careful not to use any values,
2552 which might compromise the privacy of a Principal.
- 2553 • A <Notification> element inside an end notification message MUST have a subscriptionID attribute
2554 to identify the subscription which has ended. If the subscriptionID attribute is missing, the process-
2555 ing of that <Notification> element MUST fail and a second level status code MissingSubscriptionID
2556 SHOULD be used in addition to a top level status code. If a WSC does not recognize the value of a
2557 subscriptionID attribute, the processing of that <Notification> element MUST fail and a second level
2558 status code InvalidSubscriptionID SHOULD be used in addition to a top level status code.
- 2559 • A WSP SHOULD resend an end notification for which it does not get an acknowledgement in reasonable time, if
2560 acknowledgements are used.

2561 **6.3. Examples**

2562 TBA, when a service specification using subscriptions and notifications exists (if not in reasonable time imaginary
2563 examples will be invented).

2564 7. The Schema for the DST Protocols

```
2565
2566     <?xml version="1.0" encoding="UTF-8"?>
2567 <xs:schema
2568     xmlns:xs="http://www.w3.org/2001/XMLSchema"
2569     xmlns:disco="urn:liberty:disco:2004-12"
2570     elementFormDefault="qualified"
2571     attributeFormDefault="unqualified"
2572     xmlns:sb20="urn:liberty:sb:2004-12">
2573
2574     <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
2575     <xs:import namespace="urn:liberty:disco:2004-12" schemaLocation="liberty-idwsf-disco-svc
2576 -v2.0.xsd"/>
2577     <xs:import namespace="urn:liberty:sb:20
2578 04-12" schemaLocation="liberty-idwsf-soap-binding-v2.0.xsd"/>
2579
2580     <xs:annotation>
2581         <xs:documentation>
2582 The source code in this XSD file was excerpted verbatim from:
2583
2584 Liberty ID-WSF Data Services Template Specification
2585 Version 2.0-06 Draft
2586 22 November 2004
2587
2588 Copyright (c) 2004 Liberty Alliance participants, see
2589 http://www.projectliberty.org/specs/idwsf_copyrights.html
2590
2591 NOTE: This schema must be used within the context of another schema -
2592 It is not intended to validate by itself.
2593
2594 The scheme which includes this must provide definitions for:
2595 TypeType
2596 SelectType
2597 TriggerType
2598
2599     </xs:documentation>
2600 </xs:annotation>
2601 <xs:element name="ResourceID" type="disco:ResourceIDType"/>
2602 <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>
2603 <xs:group name="ResourceIDGroup">
2604     <xs:choice>
2605         <xs:element ref="ResourceID"/>
2606         <xs:element ref="EncryptedResourceID"/>
2607     </xs:choice>
2608 </xs:group>
2609 <xs:element name="ChangeFormat">
2610     <xs:simpleType>
2611         <xs:restriction base="xs:string">
2612             <xs:enumeration value="ChangedElements"/>
2613             <xs:enumeration value="CurrentElements"/>
2614         </xs:restriction>
2615     </xs:simpleType>
2616 </xs:element>
2617 <xs:attribute name="changeFormat">
2618     <xs:simpleType>
2619         <xs:restriction base="xs:string">
2620             <xs:enumeration value="ChangedElements"/>
2621             <xs:enumeration value="CurrentElements"/>
2622             <xs:enumeration value="All"/>
2623         </xs:restriction>
2624     </xs:simpleType>
2625 </xs:attribute>
2626 <!-- Querying Data -->
2627 <xs:element name="Query" type="QueryType"/>
2628 <xs:complexType name="QueryType">
2629     <xs:sequence>
```

```
2630     <xs:group ref="ResourceIDGroup" minOccurs="0" />
2631     <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded">
2632       <xs:complexType>
2633         <xs:sequence>
2634           <xs:annotation>
2635             <xs:documentation>
2636               NOTE: The below two types (SelectType and SortType) must
2637               be defined by the schema that includes this one.
2638             </xs:documentation>
2639           </xs:annotation>
2640           <xs:element name="Select" type="SelectType" minOccurs="0" />
2641           <xs:element name="Sort" type="SortType" minOccurs="0" />
2642           <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2" />
2643         </xs:sequence>
2644         <xs:attribute name="id" type="xs:ID" />
2645         <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0" />
2646         <xs:attribute name="itemID" type="IDType" />
2647         <xs:attribute name="changedSince" type="xs:dateTime" />
2648         <xs:attribute name="count" type="xs:nonNegativeInteger" />
2649         <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0" />
2650         <xs:attribute name="setID" type="IDType" />
2651         <xs:attribute name="setReq">
2652           <xs:simpleType>
2653             <xs:restriction base="xs:string">
2654               <xs:enumeration value="Static" />
2655               <xs:enumeration value="DeleteSet" />
2656             </xs:restriction>
2657           </xs:simpleType>
2658         </xs:attribute>
2659       </xs:complexType>
2660     </xs:element>
2661     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2662   </xs:sequence>
2663   <xs:attribute name="id" type="xs:ID" />
2664   <xs:attribute name="itemID" type="IDType" />
2665 </xs:complexType>
2666 <xs:element name="QueryResponse" type="QueryResponseType" />
2667 <xs:complexType name="QueryResponseType">
2668   <xs:sequence>
2669     <xs:element ref="Status" />
2670     <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
2671       <xs:complexType>
2672         <xs:sequence>
2673           <xs:any minOccurs="0" maxOccurs="unbounded" />
2674         </xs:sequence>
2675         <xs:attribute name="id" type="xs:ID" />
2676         <xs:attribute name="itemIDRef" type="IDReferenceType" />
2677         <xs:attribute name="notSorted">
2678           <xs:simpleType>
2679             <xs:restriction base="xs:string">
2680               <xs:enumeration value="Now" />
2681               <xs:enumeration value="Never" />
2682             </xs:restriction>
2683           </xs:simpleType>
2684         </xs:attribute>
2685         <xs:attribute ref="changeFormat" />
2686         <xs:attribute name="remaining" type="xs:integer" />
2687         <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0" />
2688         <xs:attribute name="setID" type="IDType" />
2689       </xs:complexType>
2690     </xs:element>
2691     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2692   </xs:sequence>
2693   <xs:attribute name="id" type="xs:ID" />
2694   <xs:attribute name="itemIDRef" type="IDReferenceType" />
2695   <xs:attribute name="timeStamp" type="xs:dateTime" />
2696 </xs:complexType>
```

```
2697 <!-- Modifying Data -->
2698 <xs:element name="Modify" type="ModifyType"/>
2699 <xs:complexType name="ModifyType">
2700   <xs:sequence>
2701     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2702     <xs:element name="Modification" maxOccurs="unbounded">
2703       <xs:complexType>
2704         <xs:sequence>
2705           <xs:annotation>
2706             <xs:documentation>
2707               NOTE: The below SelectType must be defined by
2708               the schema that includes this one.
2709             </xs:documentation>
2710           </xs:annotation>
2711           <xs:element name="Select" type="SelectType" minOccurs="0"/>
2712           <xs:element name="NewData" minOccurs="0">
2713             <xs:complexType>
2714               <xs:sequence>
2715                 <xs:any minOccurs="0" maxOccurs="unbounded"/>
2716               </xs:sequence>
2717             </xs:complexType>
2718           </xs:element>
2719         </xs:sequence>
2720         <xs:attribute name="id" type="xs:ID"/>
2721         <xs:attribute name="itemID" type="IDType"/>
2722         <xs:attribute name="notChangedSince" type="xs:dateTime"/>
2723         <xs:attribute name="overrideAllowed" type="xs:boolean" default="0"/>
2724       </xs:complexType>
2725     </xs:element>
2726     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2727   </xs:sequence>
2728   <xs:attribute name="id" type="xs:ID"/>
2729   <xs:attribute name="itemID" type="IDType"/>
2730 </xs:complexType>
2731 <xs:element name="ModifyResponse" type="ResponseType"/>
2732 <xs:complexType name="ResponseType">
2733   <xs:sequence>
2734     <xs:element ref="Status"/>
2735     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2736   </xs:sequence>
2737   <xs:attribute name="id" type="xs:ID"/>
2738   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2739   <xs:attribute name="timeStamp" type="xs:dateTime"/>
2740 </xs:complexType>
2741 <!-- Subscribing notifications and modifying, renewing and deleting existing notifications -->
2742 <xs:element name="Subscribe" type="SubscribeType"/>
2743 <xs:complexType name="SubscribeType">
2744   <xs:sequence>
2745     <xs:group ref="ResourceIDGroup" minOccurs="0"/>
2746     <xs:element ref="Subscription" maxOccurs="unbounded"/>
2747     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2748   </xs:sequence>
2749   <xs:attribute name="id" type="xs:ID"/>
2750   <xs:attribute name="itemID" type="IDType"/>
2751   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
2752 </xs:complexType>
2753 <xs:element name="SubscribeResponse" type="SubscribeResponseType"/>
2754 <xs:complexType name="SubscribeResponseType">
2755   <xs:sequence>
2756     <xs:element ref="Status"/>
2757     <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
2758     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
2759   </xs:sequence>
2760   <xs:attribute name="id" type="xs:ID"/>
2761   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
2762   <xs:attribute name="timeStamp" type="xs:dateTime"/>
2763 </xs:complexType>
```



```
2764 <!-- Query subscriptions -->
2765 <xs:element name="QuerySubscriptions" type="QuerySubscriptionsType" />
2766 <xs:complexType name="QuerySubscriptionsType">
2767   <xs:sequence>
2768     <xs:group ref="ResourceIDGroup" minOccurs="0" />
2769     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2770   </xs:sequence>
2771   <xs:attribute name="id" type="xs:ID" />
2772   <xs:attribute name="itemID" type="IDType" />
2773 </xs:complexType>
2774 <xs:element name="Subscriptions" type="SubscriptionsType" />
2775 <xs:complexType name="SubscriptionsType">
2776   <xs:sequence>
2777     <xs:element ref="Status" />
2778     <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded" />
2779     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2780   </xs:sequence>
2781   <xs:attribute name="id" type="xs:ID" />
2782   <xs:attribute name="itemIDRef" type="IDReferenceType" />
2783 </xs:complexType>
2784 <!-- Subscription Element -->
2785 <xs:element name="Subscription">
2786   <xs:complexType>
2787     <xs:sequence>
2788       <xs:annotation>
2789         <xs:documentation>
2790           NOTE: The SelectType, TypeType, and TriggerType below
2791           must be defined by the schema that includes this one.
2792         </xs:documentation>
2793       </xs:annotation>
2794       <xs:element name="Select" type="SelectType" minOccurs="0" />
2795       <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2" />
2796       <xs:element name="NotifyTo" type="sb20:ServiceInstanceUpdateType" />
2797       <xs:element name="NotifyEndedTo" type="sb20:ServiceInstanceUpdateType" minOccurs="0" />
2798       <xs:element name="Type" type="TypeType" minOccurs="0" />
2799       <xs:element name="Trigger" type="TriggerType" minOccurs="0" />
2800       <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2801     </xs:sequence>
2802     <xs:attribute name="starts" type="xs:dateTime" />
2803     <xs:attribute name="expires" type="xs:dateTime" />
2804     <xs:attribute name="duration" type="xs:duration" />
2805     <xs:attribute name="id" type="xs:ID" />
2806     <xs:attribute name="invokeID" type="IDType" />
2807     <xs:attribute name="subscriptionID" type="IDType" />
2808     <xs:attribute name="includeData" default="Yes">
2809       <xs:simpleType>
2810         <xs:restriction base="xs:string">
2811           <xs:enumeration value="Yes" />
2812           <xs:enumeration value="No" />
2813           <xs:enumeration value="YesWithCommonAttributes" />
2814         </xs:restriction>
2815       </xs:simpleType>
2816     </xs:attribute>
2817   </xs:complexType>
2818 </xs:element>
2819
2820 <!-- Sending Notifications and Notifying about subscriptions which have ended-->
2821 <xs:element name="Notify" type="NotifyEndedType" />
2822 <xs:element name="NotifyResponse" type="NotifyEndedResponseType" />
2823 <xs:element name="Ended" type="NotifyEndedType" />
2824 <xs:element name="EndedResponse" type="NotifyEndedResponseType" />
2825 <xs:complexType name="NotifyEndedType">
2826   <xs:sequence>
2827     <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded" />
2828     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2829   </xs:sequence>
2830 <xs:attribute name="id" type="xs:ID" />
```

```
2831     <xs:attribute name="itemID" type="IDType" />
2832     <xs:attribute name="timeStamp" type="xs:dateTime" />
2833 </xs:complexType>
2834 <xs:complexType name="NotifyEndedResponseType">
2835     <xs:sequence>
2836         <xs:element ref="Status" />
2837         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
2838     </xs:sequence>
2839     <xs:attribute name="id" type="xs:ID" />
2840     <xs:attribute name="itemIDRef" type="IDReferenceType" />
2841 </xs:complexType>
2842 <!-- Notification Element -->
2843 <xs:element name="Notification">
2844     <xs:complexType>
2845         <xs:sequence>
2846             <xs:element name="Data" minOccurs="0">
2847                 <xs:complexType>
2848                     <xs:sequence>
2849                         <xs:any minOccurs="0" maxOccurs="unbounded" />
2850                     </xs:sequence>
2851                 </xs:complexType>
2852             </xs:element>
2853         </xs:sequence>
2854         <xs:attribute name="id" type="xs:ID" />
2855         <xs:attribute name="invokeID" type="IDType" />
2856         <xs:attribute name="subscriptionID" type="IDType" use="required" />
2857         <xs:attribute ref="changeFormat" />
2858         <xs:attribute name="expires" type="xs:dateTime" />
2859         <xs:attribute name="duration" type="xs:duration" />
2860         <xs:attribute name="endReason" type="xs:anyURI" />
2861     </xs:complexType>
2862 </xs:element>
2863 </xs:schema>
2864
2865
```

2866 **8. Checklist for Service Specifications**

2867 The following table provides a checklist of issues which should be addressed by individual service type specifications.
2868 Such specifications should always state which optional features of the DST they support, in addition to defining more
2869 general things such as discovery option keywords and the `SelectType` XML type used by the service type. A service
2870 specification should complete this table with the specific values and statements required by the specification.

2871 For optional features, the language specified by [\[RFC2119\]](#) MUST be used to define whether these features are
2872 available for implementations and deployments. For example, specifying that a feature 'MAY' be implemented by
2873 a WSP means that WSPs may or may not support the feature, and that WSCs should be ready to handle both cases.

2874

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.

2875

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.

2876

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.

2877

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?

2878

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.

2879

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>

2880

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 data-bbox="927 527 1377 674"> <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 <i>starts</i> attribute. It MUST be specified here, is the <i>starts</i> attribute supported or not. E.g. The <i>starts</i> 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, <i>expires</i> and <i>duration</i>, 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 <i>expires</i> and <i>duration</i> MAY be used.</p>
Support expires==starts	<p>Is it allowed to specify same time both for the <i>starts</i> and the <i>expires</i> attribute to request one notification message at a specified time. E.g. same value MAY be used both for the <i>starts</i> and the <i>expires</i> attribute.</p>
Support zero duration	<p>Is it allowed to specify the <i>duration</i> to have a zero value to have a subscription valid only for one notification. E.g. zero value for MUST NOT be used for the <i>duration</i> 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>

2881

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.

2882

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.

2883

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.

2884 **References**

2885 **Normative**

- 2886 [LibertyDisco] Beatty, John, Hodges, Jeff, Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification,"
2887 Version 2.0-02, Liberty Alliance Project (24 Nov 2004). <http://www.projectliberty.org/specs>
- 2888 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02,
2889 Liberty Alliance Project (25 November 2004). <http://www.projectliberty.org/specs>
- 2890 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, eds. "Liberty ID-WSF SOAP Binding Specification
2891 ," Version 2.0-01, Liberty Alliance Project (22 November 2004). <http://www.projectliberty.org/specs>
- 2892 [LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 2.0-01, Liberty
2893 Alliance Project (22 November 2004). <http://www.projectliberty.org/specs>
- 2894 [LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-01, Liberty
2895 Alliance Project (22 November 2004). <http://www.projectliberty.org/specs>
- 2896 [LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.1, Liberty Alliance Project
2897 (18 April 2004). <http://www.projectliberty.org/specs>
- 2898 [LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version 1.3-errata-v1.0, Liberty Alliance Project
2899 (12 Aug 2004). <http://www.projectliberty.org/specs>
- 2900 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.0, Liberty Alliance Project (12
2901 November 2003). <http://www.projectliberty.org/specs>
- 2902 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May
2903 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium
2904 <http://www.w3.org/TR/xmlschema-1/>
- 2905 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
2906 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt> [March 1997].
- 2907 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible
2908 Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium
2909 <http://www.w3.org/TR/2000/REC-xml-20001006>
- ### 2910 **Informative**
- 2911 [LibertyIDPP] Kellomaki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.0, Liberty
2912 Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 2913 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
2914 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>