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 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

Notice

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

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

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

Revision History

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

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

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

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

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

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

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

Bugs 688, 689 and 690.

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

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

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

Update schema to 2.0 (and references).

Contents

1. Overview

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

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

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

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

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

Note:

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

1.1. Notation

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

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

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

The following namespaces are used in the schema definitions:

- The prefix `xs:` stands for the W3C XML schema namespace (`http://www.w3.org/2001/XMLSchema`). [Schema1]
- The prefix `xml:` stands for the W3C XML namespace (`http://www.w3.org/XML/1998/namespace`).
- The prefix `disco:` stands for the Liberty ID-WSF Discovery Service schema namespace (`urn:liberty:disco:2003-08`). [LibertyDisco]
- The prefix `md:` stands for the Liberty Metadata schema namespace (`urn:liberty:metadata:2003-08`). [LibertyMetadata]
- The prefix `sb-ext:` stands for the Liberty ID-WSF SOAP Binding Extension schema namespace (`urn:liberty:sb:2004-04`). [LibertySOAPBinding]

The following namespaces are used in examples:

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

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

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

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

1.2. Liberty Considerations

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

2. Data Model

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

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

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

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

2.1. Guidelines for Schemas

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

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

2.2. Extending a Service

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

- An implementation MAY add new elements and attributes to the specified schema. These new elements and attributes MUST use their own XML namespace until they are adopted by the official Liberty specification and schema of the service type.

- When new features for a service are specified (e.g. new elements), new keywords SHOULD be specified for indicating the new features using the `<Option>` element (see [LibertyDisco] for more information).

- New values for enumerators MAY be specified subsequent to the release of a specification document for a specific service type. The specification for a service type MUST specify the authority for registering new official enumerators (whether that authority is the specification itself, or some external authority). For specification done by Liberty Alliance, see [LibertyReg].

- Elements defined in the XML schema for a service type MAY contain an `<xs:any>` element to support arbitrary schema extension. When the `<xs:any>` elements are in the schema, an implementation MAY support this type of extension, but is not required to. The `<xs:any>` elements SHOULD always be put inside `<Extension>` elements. If an implementation does support this type of schema extension, then it MAY register `urn:liberty:dst:can:extend` discovery option keyword. When a service holds new data, which is not defined in the schema for the service type but is stored using this kind of support for extensions, it MAY register `urn:liberty:dst:extend` discovery option keyword.

2.3. Time Values and Synchronization

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

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

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

2.4. Common Attributes

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

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

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

2.4.1. The commonAttributes Attribute Group

There are only two common attributes:

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

`modificationTime` [Optional]
: The `modificationTime` specifies the last time that the element was modified. Modification includes changing either the value of the element itself, or any sub-element. So the time of the modification MUST be propagated up all the way to the root element, when container elements have the `modificationTime` attribute. If the root element has the `modificationTime` attribute, it states the time of the latest modification. Note that a data service may have the `modificationTime` attribute used only in leaf elements or not even for those as it is optional.

2.4.2. The leafAttributes Attribute Group

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

`modifier` [Optional]
: The `modifier` is the `ProviderID` (see [LibertyMetadata]) of the service provider which last modified the data element.

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

- `urn:liberty:dst:acc:unknown`
  This means that there has been no validation, or the values are just voluntary input from the user. The `ACC` MAY be omitted in the message exchange when it has this value, as this value is equivalent to supplying no `ACC` attribute at all.

- `urn:liberty:dst:acc:incentive`
  There has been some incentive for user to supply correct input (such as a gift sent to the user in return for their input).

- `urn:liberty:dst:acc:challenge`
  A challenge mechanism has been used to validate the collected data (e.g. an email sent to address and a reply received or an SMS message sent to a mobile phone number containing a WAP URL to be clicked to complete the data collection)

- `urn:liberty:dst:acc:secondarydocuments`
  The value has been validated from secondary documents (such as the address from an electric bill).

- `urn:liberty:dst:acc:primarydocuments`
  The value has been validated from primary documents (for example, the name and identification number from a passport).

Other values are allowed for `ACC`, but this specification normatively defines usage only for the values listed above.
When the `ACC` is included in the response message, the response SHOULD be signed by the service provider hosting the data service.

`ACCTime` [Optional]
This defines the time that the value for the `ACC` attribute was given. Note that this can be different from the `modificationTime`. The `ACC` contains information that may be related to the validation of the entry. Such validation might happen later than the time the entry was made, or modified. The entry can be validated more than once.

2.4.3. The localizedLeafAttributes Attribute Group

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

`xml:lang` [Required]
This defines the language used for the value of a localized leaf element. When the `localizedLeafAttributes` attribute group is used for an element, this is a mandatory XML attribute.

`script` [Optional]
Sometimes the language does not define the writing system used. In such cases, this attribute defines the writing system in more detail. This specification defines the following values for this attribute: `urn:liberty:dst:script:kana` and `urn:liberty:dst:script:kanji`. See [LibertyReg] where to find more values and how to specify more values.

2.4.4. Individual common attributes

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

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

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

2.5. Common Data Types

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

The following type definitions are provided:

- `DSTLocalizedString`
- `DSTString`
- `DSTInteger`
- `DSTURI`
- `DSTDate`
- `DSTMonthDay`

2.6. The Schema for Common XML Attributes and Data Types

```
            <?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:disco="urn:liberty:disco:2004-12"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   xmlns:sb20="urn:liberty:sb:2004-12">

   <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
   <xs:import namespace="urn:liberty:disco:2004-12" schemaLocation="liberty-idwsf-disco
-svc-v2.0.xsd"/>
   <xs:import namespace="urn:liberty:sb:2004-12" schemaLocation="liberty-idwsf-soap-binding-v2.0.x
sd"/>

   <xs:annotation>
      <xs:documentation>
The source code in this XSD file was excerpted verbatim from:

Liberty ID-WSF Data Services Template Specification
Version 2.0-06 Draft
22 November 2004

Copyright (c) 2004 Liberty Alliance participants, see
http://www.projectliberty.org/specs/idwsf_copyrights.html

NOTE: This schema must be used within the context of another schema -
It is not intended to validate by itself.

```

```
The scheme which includes this must provide definitions for:
TypeType
SelectType
TriggerType
SortType

      </xs:documentation>
   </xs:annotation>
   <xs:element name="ResourceID" type="disco:ResourceIDType"/>
   <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>
   <xs:group name="ResourceIDGroup">
      <xs:choice>
         <xs:element ref="ResourceID"/>
         <xs:element ref="EncryptedResourceID"/>
      </xs:choice>
   </xs:group>
   <xs:element name="ChangeFormat">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ChangedElements"/>
            <xs:enumeration value="CurrentElements"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:element>
   <xs:attribute name="changeFormat">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ChangedElements"/>
            <xs:enumeration value="CurrentElements"/>
            <xs:enumeration value="All"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <!-- Querying Data -->
   <xs:element name="Query" type="QueryType"/>
   <xs:complexType name="QueryType">
      <xs:sequence>
         <xs:group ref="ResourceIDGroup" minOccurs="0"/>
         <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:annotation>
                     <xs:documentation>
                        NOTE: The below two types (SelectType and SortType) must
                        be defined by the schema that includes this one.
                     </xs:documentation>
                  </xs:annotation>
                  <xs:element name="Select" type="SelectType" minOccurs="0"/>
                  <xs:element name="Sort" type="SortType" minOccurs="0"/>
                  <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:ID"/>
               <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0"/>
               <xs:attribute name="itemID" type="IDType"/>
               <xs:attribute name="changedSince" type="xs:dateTime"/>
               <xs:attribute name="count" type="xs:nonNegativeInteger"/>
               <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0"/>
               <xs:attribute name="setID" type="IDType"/>
               <xs:attribute name="setReq">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:enumeration value="Static"/>
                        <xs:enumeration value="DeleteSet"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:attribute>
            </xs:complexType>
```

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

   <xs:element name="QueryResponse" type="QueryResponseType"/>
   <xs:complexType name="QueryResponseType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:any minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:ID"/>
               <xs:attribute name="itemIDRef" type="IDReferenceType"/>
               <xs:attribute name="notSorted">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:enumeration value="Now"/>
                        <xs:enumeration value="Never"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:attribute>
               <xs:attribute ref="changeFormat"/>
               <xs:attribute name="remaining" type="xs:integer"/>
               <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0"/>
               <xs:attribute name="setID" type="IDType"/>
            </xs:complexType>
         </xs:element>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
      <xs:attribute name="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
   <!-- Modifying Data -->
   <xs:element name="Modify" type="ModifyType"/>
   <xs:complexType name="ModifyType">
      <xs:sequence>
         <xs:group ref="ResourceIDGroup" minOccurs="0"/>
         <xs:element name="Modification" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:annotation>
                     <xs:documentation>
                        NOTE: The below type must be defined by
                        the schema that includes this one.
                     </xs:documentation>
                  </xs:annotation>
                  <xs:element name="Select" type="SelectType" minOccurs="0"/>
                  <xs:element name="NewData" minOccurs="0">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:any minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
               <xs:attribute name="id" type="xs:ID"/>
               <xs:attribute name="itemID" type="IDType"/>
               <xs:attribute name="notChangedSince" type="xs:dateTime"/>
               <xs:attribute name="overrideAllowed" type="xs:boolean" default="0"/>
            </xs:complexType>
         </xs:element>
```

```
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemID" type="IDType"/>
   </xs:complexType>
   <xs:element name="ModifyResponse" type="ResponseType"/>
   <xs:complexType name="ResponseType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
      <xs:attribute name="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
   <!-- Subscribing notifications and modifying, renewing and deleting existing notifications -->
   <xs:element name="Subscribe" type="SubscribeType"/>
   <xs:complexType name="SubscribeType">
      <xs:sequence>
         <xs:group ref="ResourceIDGroup" minOccurs="0"/>
         <xs:element ref="Subscription" maxOccurs="unbounded"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemID" type="IDType"/>
      <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
   </xs:complexType>
   <xs:element name="SubscribeResponse" type="SubscribeResponseType"/>
   <xs:complexType name="SubscribeResponseType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
      <xs:attribute name="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
   <!-- Query subscriptions -->
   <xs:element name="QuerySubscriptions" type="QuerySubscriptionsType"/>
   <xs:complexType name="QuerySubscriptionsType">
      <xs:sequence>
         <xs:group ref="ResourceIDGroup" minOccurs="0"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemID" type="IDType"/>
   </xs:complexType>
   <xs:element name="Subscriptions" type="SubscriptionsType"/>
   <xs:complexType name="SubscriptionsType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   </xs:complexType>
   <!-- Subscription Element -->
   <xs:element name="Subscription">
      <xs:complexType>
         <xs:sequence>
            <xs:annotation>
               <xs:documentation>
                  NOTE: The below types (SelectType, TypeType, and TriggerType)
                  must be defined by the schema that includes this one.
               </xs:documentation>
```

```
            </xs:annotation>
            <xs:element name="Select" type="SelectType" minOccurs="0"/>
            <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
            <xs:element name="NotifyTo" type="sb20:ServiceInstanceUpdateType"/>
            <xs:element name="NotifyEndedTo" type="sb20:ServiceInstanceUpdateType" minOccurs="0"/>
            <xs:element name="Type" type="TypeType" minOccurs="0"/>
            <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="starts" type="xs:dateTime"/>
         <xs:attribute name="expires" type="xs:dateTime"/>
         <xs:attribute name="duration" type="xs:duration"/>
         <xs:attribute name="id" type="xs:ID"/>
         <xs:attribute name="invokeID" type="IDType"/>
         <xs:attribute name="subscriptionID" type="IDType"/>
         <xs:attribute name="includeData" default="Yes">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="Yes"/>
                  <xs:enumeration value="No"/>
                  <xs:enumeration value="YesWithCommonAttributes"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <!-- Sending Notifications and Notifying about subscriptions which have ended-->
   <xs:element name="Notify" type="NotifyEndedType"/>
   <xs:element name="NotifyResponse" type="NotifyEndedResponseType"/>
   <xs:element name="Ended" type="NotifyEndedType"/>
   <xs:element name="EndedResponse" type="NotifyEndedResponseType"/>
   <xs:complexType name="NotifyEndedType">
      <xs:sequence>
         <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemID" type="IDType"/>
      <xs:attribute name="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
   <xs:complexType name="NotifyEndedResponseType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   </xs:complexType>
   <!-- Notification Element -->
   <xs:element name="Notification">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Data" minOccurs="0">
               <xs:complexType>
                  <xs:sequence>
                     <xs:any minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="id" type="xs:ID"/>
         <xs:attribute name="invokeID" type="IDType"/>
         <xs:attribute name="subscriptionID" type="IDType" use="required"/>
         <xs:attribute ref="changeFormat"/>
         <xs:attribute name="expires" type="xs:dateTime"/>
         <xs:attribute name="duration" type="xs:duration"/>
         <xs:attribute name="endReason" type="xs:anyURI"/>
```

```
      </xs:complexType>
   </xs:element>
</xs:schema>
```

3. Message Interface

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

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

Table 1. Request/Response

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

Table 2. Notification/Acknowledgement

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

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

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

3.1. Status and Fault Reporting

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

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

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

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

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

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

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

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

3.1.1. <Status> element

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

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

- `ActionNotAuthorized`
- `ActionNotSupported`
- `AllReturned`
- `ChangeHistoryNotSupported`
- `ChangedSinceReturnsAll`
- `DataTooLong`
- `ExistsAlready`

- ExtensionNotSupported
- Failed
- FormatNotSupported
- InvalidData
- InvalidExpires
- InvalidEndedTo
- InvalidEndpoint
- InvalidResourceID
- InvalidSelect
- InvalidSetID
- InvalidSetReq
- InvalidSort
- InvalidSubscriptionID
- MissingCredentials
- MissingDataElement
- MissingEndpointElement
- MissingExpiration
- MissingItemID
- MissingNewDataElement
- MissingNotifyToElement
- MissingResourceIDElement
- MissingSecurityMechIDElement
- MissingSelect
- MissingSubscriptionID
- ModifiedSince
- NewOrExisting
- NoMoreElements
- NoMultipleAllowed
- NoMultipleResources
- 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).

`OK`

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

`Partial`

The value `Partial` means that the processing has succeeded only partially and partially failed, e.g. in the processing of a `<Query>` element some `<QueryItem>` element has been processed successfully, but the processing of some other `<QueryItem>` elements has failed. When the value `Partial` is used for the `code` attribute of the top level `<Status>` element, the top level `<Status>` element MUST have second level `<Status>` element(s) to indicate the failed part(s) of a `[RequestElement]` element or a `[NotificationElement]` element. The processing of the part(s) not referred to by any of the second level `<Status>` elements MUST have succeeded. A WSP MUST NOT use the value `Partial`, if it has not processed the whole `[RequestElement]` element or `[NotificationElement]` element.

A WSP MUST NOT use the value `Partial` in case of modification requests, when a failed `<Modification>` element didn't have a valid `itemID` attribute, i.e. a WSP is not able to indicate the failed `<Modification>` element. In those cases a WSP MUST use the value `Failed` and anything changed based on the already processed part MUST be rolled back.

A WSP MAY also choose to fail completely another type of `[RequestElement]`, when only a part of it has failed, if the failed part does not have a valid `itemID` attribute. When ever the top level value `Failed` is used instead of `Partial` due to one or more missing `itemID` attribute(s), the second level status code `MissingItemID` MUST be used in addition to any other second level status code.

In some cases the most descriptive second level status code may not be used as it e.g. might compromise the privacy of a Principal. In those cases, when the second level status code must be used to indicate the failed parts in a case of a partial failure, the value `UnspecifiedError` MUST be used for the second level status code.

`Failed`

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

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

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

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

3.2. Linking with ids

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

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

See the definitions and the processing rules of the protocol elements and the processing rules in Section 3.3 for more detailed information.

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

3.3. Resources

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

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

The ResourceIDGroup schema is shown below:

```
<xs:element name="ResourceID" type="disco:ResourceIDType"/>
<xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>
<xs:group name="ResourceIDGroup">
   <xs:choice>
      <xs:element ref="ResourceID"/>
      <xs:element ref="EncryptedResourceID"/>
   </xs:choice>
</xs:group>
```

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

Note:

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

3.3.1. Common processing rules

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

- A receiver of a message (WSP for request messages and WSC for notification messages) MUST support one [RequestElement] element inside a request message and one [NotificationElement] element inside a notification message and SHOULD support multiple. If a receiver supports only one [RequestElement] or [NotificationElement] element inside a request/notification message and the message contains multiple [RequestElement] or [NotificationElement] elements, the processing of the whole message MUST fail and a status code indicating the failure MUST be returned in the response. A more detailed status code with the value NoMultipleResources SHOULD be returned in addition to the top level status code as it is not possible to handle multiple resources in one message. If a WSP supports accessing multiple resources, it MAY register urn:liberty:dst:multipleResources discovery option keyword.

- If a request message contains multiple [RequestElement] elements or a notification message contains multiple [NotificationElement] elements, the sender MUST add itemID attributes for each [RequestElement]/[NotificationElement] element. The receiver MUST link the [ResponseElement]/<AcknowledgementElement> elements it is sending to corresponding [RequestElement]/[NotificationElement] elements it received using the itemIDRef attributes, if there were itemID attributes in the [RequestElement]/[NotificationElement] elements and there were multiple [RequestElement]/[NotificationElement] elements in the request message.
  If multiple [RequestElement] or [NotificationElement] elements are used in a message, a receiver MUST discard any [RequestElement] or [NotificationElement] element not having a valid itemID attribute. For that [RequestElement] or [NotificationElement] element no <ResponseElement> or [AcknowledgementElement] element is returned as it is not possible to indicate for which [RequestElement] or [NotificationElement] element that element is returned.
  The itemIDRef attribute in a [ResponseElement]/<AcknowledgementElement> element MUST have the same value as the itemID attribute in the corresponding [RequestElement]/[NotificationElement] element.

- If the processing of a [RequestElement]/[NotificationElement] fails for some reason, any other [RequestElement]/[NotificationElement] elements included in the same message SHOULD be processed normally, as if the error had not occurred as different [RequestElement]/[NotificationElement] elements inside the same message are usually independent. When processing of a [RequestElement]/[NotificationElement] fails completely, the top level status code Failed MUST be used to indicate the failure and a more detailed status code SHOULD be used to indicate more detailed status information. If it is possible and allowed by the service specification to process successfully independent part(s) of the [RequestElement]/[NotificationElement] the top level status code Partial MUST be used as specified earlier. A successful request/notification MUST be indicated using the top level status code OK. Note: even with top level status code OK second level status codes MAY be used.

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

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

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

3.4. <Select> element

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

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

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

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

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

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

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

3.4.1. Common processing rules

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

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

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

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

3.5. The timeStamp Attribute

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

3.5.1. Common processing rules

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

3.6. The <Extension> Element

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

3.7. General error handling

This subchapter defines processing rules for some general error cases.

A WSP may not support all different type of requests:

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

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

- If the processing takes too long (for example some back-end system is not responding fast enough) the second level status code `TimeOut` SHOULD be used to indicate this, when the request is not processed due to a WSP internal time out. The WSP defines how long it tries to process before giving up and returning the `TimeOut` status code. Note that [LibertySOAPBinding] specifies a header block which a WSC may use to define threshold for timeout, but that is different functionality and the processing rules for that are specified in [LibertySOAPBinding].

- Other error conditions than those listed in this specification and in service specifications may occur. There are two status codes defined for those cases. For cases a WSP (or WSC receiving a notification) can handle normally but for which there is no status code specified, the second level status code `UnspecifiedError` SHOULD be used. For totally unexpected cases the second level status code `UnexpectedError` SHOULD be used.

4. Querying Data

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

4.1. The <Query> Element

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

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

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

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

```
<xs:element name="ChangeFormat">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="ChangedElements"/>
         <xs:enumeration value="CurrentElements"/>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

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

`includeCommonAttributes` [Optional]
: The `includeCommonAttributes` specifies what kind of response is requested. The default value is *False*, which means that only the data specified in the service definition is returned. If the common attributes specified for container and leaf elements in this document are also needed, then this attribute must be given the value *True*. If the `id` attribute is used for distinguishing similar elements from one other by the service, it MUST always be returned, even if the `includeCommonAttributes` is *False*.
The `xml:lang` and `script` attributes are always returned when they exist.

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

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

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

4.1.1. Pagination

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

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

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

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

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

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

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

4.1.2. The schema for the <Query> element

```
<xs:element name="Query" type="QueryType"/>
<xs:complexType name="QueryType">
   <xs:sequence>
      <xs:group ref="ResourceIDGroup" minOccurs="0"/>
      <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded">
```

```
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Select" type="SelectType" minOccurs="0"/>
                  <xs:element name="Sort" type="SortType" minOccurs="0"/>
                  <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:ID"/>
               <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0"/>
               <xs:attribute name="itemID" type="IDType"/>
               <xs:attribute name="changedSince" type="xs:dateTime"/>
               <xs:attribute name="count" type="xs:nonNegativeInteger"/>
               <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0"/>
               <xs:attribute name="setID" type="IDType"/>
               <xs:attribute name="setReq">
                  <xs:simpleType>
                     <xs:restriction base="xs:string">
                        <xs:enumeration value="Static"/>
                        <xs:enumeration value="DeleteSet"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:attribute>
            </xs:complexType>
         </xs:element>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemID" type="IDType"/>
   </xs:complexType>
```

4.2. The <QueryResponse> Element

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

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

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

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

```
   <xs:attribute name="changeFormat">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ChangedElements"/>
            <xs:enumeration value="CurrentElements"/>
            <xs:enumeration value="All"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
```

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

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

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

The schema for `<QueryResponse>` is below:

```
<xs:element name="QueryResponse" type="QueryResponseType"/>
<xs:complexType name="QueryResponseType">
   <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:sequence>
               <xs:any minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID"/>
            <xs:attribute name="itemIDRef" type="IDReferenceType"/>
            <xs:attribute name="notSorted">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:enumeration value="Now"/>
                     <xs:enumeration value="Never"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:attribute>
            <xs:attribute ref="changeFormat"/>
            <xs:attribute name="remaining" type="xs:integer"/>
            <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0"/>
            <xs:attribute name="setID" type="IDType"/>
         </xs:complexType>
      </xs:element>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   <xs:attribute name="timeStamp" type="xs:dateTime"/>
</xs:complexType>
```

4.3. Processing Rules

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

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

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

- If the <Query> contains multiple <QueryItem> elements, the WSC MUST add itemID attributes to each <QueryItem> element. The WSP MUST link the <Data> elements to corresponding <QueryItem> elements using the itemIDRef attributes, if there were itemID attributes in the <QueryItem> elements and there were multiple <QueryItem> elements in the <Query>. The itemIDRef attribute in a <Data> element MUST have the same value as the itemID attribute in the corresponding <QueryItem> element.

- If processing of a <QueryItem> fails, any remaining unprocessed <QueryItem> elements SHOULD NOT be processed. The data for the already processed <QueryItem> elements SHOULD be returned in the response message and the status code MUST indicate the failure to completely process the whole <Query>. A more detailed status SHOULD be used in addition to the top level status code to indicate the reason for failing to process the first failed <QueryItem>.

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

- If there is no changedSince attribute in the <QueryItem> element and the <Select> requests valid data element(s), but there are no values, the WSP MUST NOT return any <Data> element for that <QueryItem>.

- If the <Select> requests multiple data elements, the WSP MUST return all of those data elements inside the <Data> element corresponding to the containing <QueryItem>.

A WSC may request that the data returned is sorted.

- When the <Sort> element is included in a <QueryItem> element, the data returned inside a <Data> element SHOULD be sorted according to the criteria given in the <Sort> element. If a WSP doesn't support sorting, it SHOULD return the requested data unsorted. When the data is returned unsorted, the notSorted attribute MUST be used in the <Data> element containing the unsorted data. A WSP MAY also choose to fail to process the <QueryItem>, if it does not support sorting. In that case the second level status code SortNotSupported SHOULD be used in addition to the top level status code. A WSP may also register discovery option keyword urn:liberty:dst:noSorting, if the sorting has been specified for the service type, but the WSP doesn't support it.

- If the content of the <Sort> element is not according to service specifications, a WSP SHOULD return the requested data unsorted. When the data is returned unsorted, the notSorted attribute MUST be used in the <Data> element containing the unsorted data and the second level status code InvalidSort SHOULD also be used. A WSP MAY also choose to fail to process the <QueryItem>, if the content of the <Sort> element is not according to service specifications. In this kind of a case the second level status code InvalidSort SHOULD be used in addition to the top level status code. If the the content of the <Sort> element is valid, but a WSP does not support the requested type of sorting, it SHOULD return the requested data unsorted. When the data is returned unsorted, the notSorted attribute MUST be used in the <Data> element containing the unsorted data. A WSP MAY also choose to fail to process of the <QueryItem>, if it does not support the requested type of sorting. It SHOULD use the second level status code RequestedSortingNotSupported in addition to the top level status code.

- When the notSorted attribute is used, it MUST have the value Now, when a WSP supports sorting, but not the requested type or the content of the <Sort> element was invalid. The notSorted attribute MUST have the value Never, when a WSP does not support sorting at all.

A WSC may want to receive the data in smaller sets instead of getting all the data at once, when there can be many elements with the same name. A WSC indicates this using either or both of the attributes count and offset in a <QueryItem> element, when the <Select> addresses a set of elements all having the same name. The number of elements inside this set may be restricted further by other parameters. Also access rights and policies may reduce the set of elements a WSC is allowed to get.

- A WSP MUST always follow the same ordering procedure, when the <Select> and <Sort> elements have the same values and either or both of attributes count and offset are used in the <QueryItem> element. This is 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 previously returned items with five new items.

- When either or both of the attributes count and offset is used in a <QueryItem> element and a WSP doesn't support pagination, the processing of whole <QueryItem> element MUST fail and the second level status code PaginationNotSupported SHOULD be used in addition to the top level status code. A WSP may support pagination, but not for the requested elements. In such a case the processing of whole <QueryItem> element MUST fail and the second level status code RequestedPaginationNotSupported SHOULD be used in addition to the top level status code. If a WSP doesn't support pagination at all, it MAY register the discovery option keyword urn:liberty:dst:noPagination to indicate this.

- When the count attribute is included in a <QueryItem> element, the corresponding <Data> element in the <QueryResponse> MUST NOT contain more elements addressed with the value of the <Select> element than specified by the count attribute. A WSP MAY return a smaller number of elements of the same name that requested by a WSC. If the count attribute has the value zero, the WSP MUST NOT return any data elements inside the <Data> element. This count="0" may be used for querying the number of remaining elements starting from the specified offset, e.g. from offset zero, i.e. the total number of the elements addressed by the <Select> element. When the count attribute is not used in a <QueryItem> element, it means that the WSC requests for all data specified by other parameters like the <Select> element starting from the specified offset. As the default value for the offset attribute is zero, the case when neither of the attributes offset or count is not present reduces to a normal query.

- When pagination is requested by a WSC, the elements inside a <Data> element MUST be in the ascending order of their offsets. The first element MUST have the offset specified by the offset attribute in the <QueryItem> element. The <Data> element MUST have both attributes nextOffset and remaining. The nextOffset attribute MUST have the offset of the first element not returned in the response. The value of the remaining attribute MUST define how many elements there are left starting from the value of the nextOffset, if a WSP knows that (e.g. that information might not be available from a backend system). If WSP does not know the exact value, it MUST use the value -1 for the remaining attribute until it knows the value or there is no data left (remaining="0"). When remaining="-1", a WSC must make new requests until remaining="0", if it wants to get all the data.

- When the setReq attribute is included in a <QueryItem> element and has the value Static, the WSP SHOULD return the setID attribute to the requesting WSC and process <QueryItem> elements later having this setID based on the data the WSP has at the time, when the value for the setID was created. If a WSP receives a <QueryItem> element having the setReq attribute and does not support static sets for the requested data or not at all, the processing of the <QueryItem> element MUST fail and a second level status code StaticNotSupported SHOULD be used in addition to the top level status code. If a WSP doesn't support static sets at all, it MAY register the discovery option keyword urn:liberty:dst:noStatic.

- When the setID attribute is included in a request, the following parameters MUST NOT be used in a <QueryItem> element: the <Select> element, the <Sort> element, the changedSince attribute or the includeCommonAttributes attribute. The requests are made from an earlier defined static set and the count and the offset attributes are used to define, what is requested from that set. If any of the mentioned parameters is present, when the setID attribute is used, it is unclear what a WSC wants and the processing of the whole <QueryItem> MUST fail and a second level status code SetOrNewQuery SHOULD be used in addition to the top level status code.

- When the setID attribute is included in a <QueryItem> element and has a valid value, the <Data> element in the response MUST always have the setID attribute.

- When a static set is created, the requesting WSC MUST query all the data it needs from this set as soon as possible and delete the static set immediately after this using `setReq="DeleteSet"`. A WSP MAY also delete the static 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 non-existing static set, the processing of the whole `<QueryItem>` MUST fail and the second level status code `InvalidSetID` SHOULD be used in addition to the top level status code.

- The `setReq="Static"` and the `setID` attribute MUST NOT be used simultaneously in a `<QueryItem>` element. If they are used, the WSP MUST ignore the `setReq="Static"` and process the `<QueryItem>` element like the `setReq` attribute would not be present.

- If the `setReq` attribute has some other value than `Static` or `DeleteSet`, the processing of the whole `<QueryItem>` element must fail and a second level status code `InvalidSetReq` SHOULD be used in addition to the top level status code.

Even when the requested data exists, it should be noted that access and privacy policies specified by the resource owner may cause the request to result in data not being returned to the requester.

- When a WSP processes a `<QueryItem>`, it MUST check whether the resource owner (the Principal, for example) has given consent to return the requested information. To be able to check WSC specific access rights, the WSP MUST authenticate the WSC (see [LibertySecMech] and [LibertyMetadata]). The WSP MUST also check that any usage directive given in the request is acceptable based on the usage directives defined by the resource owner (see [LibertySOAPBinding]). If either check fails for any piece of the requested data, the WSP MUST NOT return that piece of data. Note that there can be consent for returning some data element, but not its attributes. E.g. a resource owner might not want to release the `modifier` attribute, if she does not want to reveal information about which services she uses. The data for which there is no consent from the resource owner MUST be handled as if there was no data. The WSP MAY try to get consent from the resource owner while processing the request, e.g. by using an interaction service (see [LibertyInteract]). A WSP might check the access rights and policies in usage directives at a higher level, before getting to DST processing and MAY, in this case, just return an ID-* Fault Message [LibertySOAPBinding] without processing the `<Query>` element at all, if the requesting WSC is not allowed to access data.

It is possible to query changes since a specified time using the `changedSince` attribute. The following rules specify how this works:

- If the `<QueryItem>` element contains the `changedSince` attribute, the WSP SHOULD return only those elements addressed by the `<Select>` which have been modified since the time specified in the `changedSince` attribute. There are two different formats, in which the changed data can be returned. A WSC SHOULD indicate using the `<ChangeFormat>` element the format it prefers and also, if it understands the other format. The two formats are `ChangedElements` and `CurrentElements`. If a service specification doesn't specify anything else the value `ChangedElements` MUST be used as a default value as it is compatible with the format used in the version 1.0 of the Data Services Template.

- A WSP MUST ignore the `<ChangeFormat>` element, if the `changedSince` attribute is not used in the same `<QueryItem>` element. A WSP MUST NOT use a format, which a WSC does not understand. Note that format `ChangedElements`, has the format `All` as a fallback solution, when a WSP doesn't have all the needed change history information. Also if a WSP doesn't support requesting only changed data, it returns all data.

- A `<QueryItem>` element MAY contain two `<ChangeFormat>` element with different values. A WSP SHOULD use the format specified by the first `<ChangeFormat>` element, but, if it does not support that format, it MAY use the format specified by the second `<ChangeFormat>` element.

- If a WSP does not support the format a WSC is requesting to be used, the processing of the <QueryItem> MUST fail and the second level status code FormatNotSupported SHOULD be used in addition to the top level status code.

- If a WSC requests the ChangedElements format and a WSP supports it, the WSP SHOULD return only the changed information. if some element has been deleted, a WSP SHOULD return an empty element to indicate the deletion (<ElementName/>). The only allowed exception to this is that the WSP does not have enough history information available to be able to return only the changed information. In that case it MUST use format All and return all current elements with their values even if those have not changed since the specified time.

- If a WSC requests the CurrentElements format and a WSP supports it, the WSP SHOULD return only the currently existing elements. It SHOULD return an empty element, if an element has not changed to indicate that no change has happened (<ElementName/>).

- Note: as empty elements are used to indicate either deleted or not changed elements depending on the used format, the formats CurrentElements and ChangedElements do not work well, if the data hosted by a service may contain empty elements. In those cases a service should either use only format All or always have some attribute(s) for the otherwise empty elements.

- If a WSC has used the <ChangeFormat> element in a request, a WSP MUST use the changeFormat attribute in the response to indicate, which format is used. A WSP MUST not use the the changeFormat attribute in a response, if the <ChangeFormat> element was not used in the corresponding request so the processing stays version 1.0 compatible, when the <ChangeFormat> element is not used.

- If there can be multiple elements with same name, the id attribute or some other attribute used to distinguish the elements from each other MUST be included (e.g. in case of an ID-SIS Personal Profile service the following empty element could be returned <AddressCard id="tr7632q"/> to indicate a deleted or not changed <AddressCard> depending on the used format). If the value of the id attribute or some other attribute used for distinguishing elements with same name is changed, the WSP MUST consider this as a case, in which the element with the original value of the distinguishing attribute is deleted and a new one with the new value of the distinguishing attribute is created. To avoid this, a WSP MAY refuse to accept modifications of a distinguishing attribute and MAY require that an explicit deletion of the element is done and a new one created.

- If the elements addressed by the <Select> have some values, but there has been no changes since the time specified in the changedSince attribute, the WSP MUST return empty <Data> element (<Data/>), when it returns the changes properly. This empty <Data> element indicates that no changes have occurred. There might be cases in which the WSP is not able to return changes properly, see later processing rules. Please note that in cases that have no values, no <Data> element is returned to indicate this. So empty <Data> element has different semantics than missing <Data> element.

- If the <QueryItem> element contains the changedSince attribute and a WSP is not keeping track of modification times, it SHOULD process the <QueryItem> element as there would be no changedSince attribute, and indicate this in the response using the second level status code ChangedSinceReturnsAll. This is not considered a failure and the rest of the <QueryItem> elements MUST be processed. Also it might be that a WSP does not have a full change history and so for some queries, it is not possible to find out, which changes occurred after the specified time. As processing with access rights and policy in place might be quite complex, a WSP might sometimes process the query for changes properly and sometime process it as if there were no changedSince attribute. In those cases, when a WSP returns all current values, it SHOULD indicate this with the second level status code AllReturned and, if the <ChangeFormat> element was used in the request, the changeFormat attribute with the value All SHOULD be used. This is also not considered a failure and the rest of the <QueryItem> elements MUST be processed. Please note that the status code AllReturned differs from the status code ChangedSinceReturnsAll, as ChangedSinceReturnsAll means that the WSP never processes the changedSince attribute properly. A WSP MUST use either AllReturned or ChangedSinceReturnsAll as the second level status code, when it returns data, but does not process the changedSince attribute properly, i.e. returns only the changes. If a WSP will not process the <QueryItem> elements with a changedSince attribute

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

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

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

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

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

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

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

4.4. Examples

Please note that as the first examples are based on the [LibertyIDPP] all elements defined in this specifications are also in the namespace defined in [LibertyIDPP].

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

```
<pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:QueryItem itemID="name">
      <pp:Select>/pp:PP/pp:CommonName</pp:Select>
   </pp:QueryItem>
   <pp:QueryItem itemID="home">
      <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType="urn: liberty:id-sis-pp:addrType:
home"]</pp:Select>
   </pp:QueryItem>
</pp:Query>
```

This query may generate the following response:

```
<pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:Status code="OK"/>
   <pp:Data itemIDRef="name">
      <pp:CommonName>
         <pp:CN>Zita Lopes</pp:CN>
         <pp:AnalyzedName nameScheme="firstlast">
            <pp:FN>Zita</pp:FN>
            <pp:SN>Lopes</pp:SN>
            <pp:PersonalTitle>Dr.</pp:PersonalTitle>
         </pp:AnalyzedName>
         <pp:AltCN>Maria Lopes</pp:AltCN>
         <pp:AltCN>Zita Ma Lopes</pp:AltCN>
      </pp:CommonName>
   </pp:Data>
   <pp:Data itemIDRef="home">
      <pp:AddressCard id='9812'>
         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
         <pp:Address>
            <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way North</pp:PostalAddress>
            <pp:PostalCode>98503-2341</pp:PostalCode>
            <pp:L>Olympia</pp:L>
            <pp:ST>wa</pp:ST>
            <pp:C>us</pp:C>
         </pp:Address>
      </pp:AddressCard>
   </pp:Data>
</pp:QueryResponse>
```

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

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

```
   <pp:Status code="OK"/>
   <pp:Data itemIDRef="home">
      <pp:AddressCard id='9812'>
         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
         <pp:Address>
            <pp:C>us</pp:C>
         </pp:Address>
      </pp:AddressCard>
   </pp:Data>
</pp:QueryResponse>
```

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

```
<pp:QueryResponse  xmlns:pp="urn:liberty:id-sis-pp:2003-08"
timeStamp="2003-02-28T12:10:12Z">
   <pp:Status code="OK"/>
</pp:QueryResponse>
```

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

```
<pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:QueryItem includeCommonAttributes="True">
      <pp:Select>/pp:PP/pp:LegalIdentity/pp:VAT</pp:Select>
   </pp:QueryItem>
</pp:Query>
```

This query may generate the following response:

```
<pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
id="12345" timeStamp="2003-05-28T23:10:12Z">
   <pp:Status code="OK"/>
   <pp:Data>
      <pp:VAT modifier="http://www.accountingservices.com"
         modificationTime="2003-04-25T15:42:11Z"
         attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">
        <pp:IDValue modifier="http://www.accountingservices.com"
            modificationTime="2003-04-25T15:42:11Z"
            attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">50267
7123</IDValue>
         <pp:IDType modifier="http://www.accountingservices.com"
            modificationTime="2003-03-12T09:12:09Z"
            attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">urn:l
iberty:altIDType:itcif</IDType>
      </pp:VAT>
   </pp:Data>
</pp:QueryResponse>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   . . .
</ds:Signature>
```

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

```
<pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:QueryItem changedSince="2003-02-28T12:10:12Z">
      <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
   </pp:QueryItem>
</pp:Query>
```

This query can generate following response:

```
<pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
timeStamp="2003-05-30T16:10:12Z">
   <pp:Status code="OK"/>
   <pp:Data>
      <pp:AddressCard id='9812'>
         <pp:Address>
            <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
         </pp:Address>
      </pp:AddressCard>
      <pp:AddressCard id='w1q2'/>
   </pp:Data>
</pp:QueryResponse>
```

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

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

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

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

```
<pp:Query xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.example.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:QueryItem changedSince="2003-02-28T12:10:12Z">
      <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
      <pp:ChangeFormat>CurrentElements</pp:ChangeFormat>
   </pp:QueryItem>
</pp:Query>
```

All the current elements are returned in the response:

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

```
   <pp:Data changeFormat="CurrentElements">
      <pp:AddressCard id='9812'>
         <pp:Address>
            <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
            <pp:PostalCode/>
            <pp:L/>
            <pp:ST/>
            <pp:C/>
         </pp:Address>
      </pp:AddressCard>
   </pp:Data>
</pp:QueryResponse>
```

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

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

```
<pp:QueryResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
timeStamp="2003-05-30T16:10:12Z">
   <pp:Status code="OK>
      <Status code="ChangeSinceReturnsAll"/>
   </pp:Status>
   <pp:Data changeFormat="All">
      <pp:AddressCard id='9812'>
         <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
         <pp:Address>
            <pp:PostalAddress>2891 Madrona Beach Way North</pp:PostalAddress>
            <pp:PostalCode>98503-2341</pp:PostalCode>
            <pp:L>Olympia</pp:L>
            <pp:ST>wa</pp:ST>
            <pp:C>us</pp:C>
         </pp:Address>
      </pp:AddressCard>
   </pp:Data>
</pp:QueryResponse>
```

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

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

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

```
<ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
   <ads:ResourceID>http://provider.example.com/ohj243hj24</ads:ResourceID>
   <ads:QueryItem count="10">
      <ads:Select>Pointing to the AddressCards</ads:Select>
      <ads:Sort>Requesting sorting by the City</ads:Sort>
   </ads:QueryItem>
</ads:Query>
```

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

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

Then it queries the next ten starting from offset 10:

```
<ads:Query xmlns:ads="http://www.example.com/2010/12/AddressService">
   <ads:ResourceID>http://provider.com/ohj243hj24</ads:ResourceID>
   <ads:QueryItem count="10" offset="10">
      <ads:Select>Pointing to the AddressCards</ads:Select>
      <ads:Sort>Requesting sorting by the City</ads:Sort>
   </ads:QueryItem>
</ads:Query>
```

and gets those

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

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

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

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

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

Finally the WSC A fetches the last address card.

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

and gets the 31st address card from offset 30.

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

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

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

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

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

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

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

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

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

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

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 ten address cards.

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

So the WSC A gets the last ten address card.

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

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

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

And the WSP acknowledges the request.

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

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

5. Modifying Data

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

5.1. <Modify> element

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

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

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

The schema for <Modify>

```
<xs:element name="Modify" type="ModifyType"/>
<xs:complexType name="ModifyType">
   <xs:sequence>
      <xs:group ref="ResourceIDGroup" minOccurs="0"/>
      <xs:element name="Modification" maxOccurs="unbounded">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="Select" type="SelectType" minOccurs="0"/>
               <xs:element name="NewData" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:any minOccurs="0" maxOccurs="unbounded"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID"/>
            <xs:attribute name="itemID" type="IDType"/>
            <xs:attribute name="notChangedSince" type="xs:dateTime"/>
            <xs:attribute name="overrideAllowed" type="xs:boolean" default="0"/>
         </xs:complexType>
      </xs:element>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
```

```
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemID" type="IDType"/>
   </xs:complexType>
```

5.2. <ModifyResponse> element

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

The schema for <ModifyResponse>

```
   <xs:element name="ModifyResponse" type="ResponseType"/>
   <xs:complexType name="ResponseType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
      <xs:attribute name="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
```

5.3. Processing Rules

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

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

- A WSP MUST support one <Modification> element inside a <Modify> and SHOULD support multiple. If the <Modify> contains multiple <Modification> elements and the WSP supports only one <Modification> element inside a <Modify>, the processing of the whole <Modify> MUST fail and a status code indicating failure MUST be returned in the response. The value NoMultipleAllowed SHOULD be used for the second level status code. If a WSP supports multiple <Modification> element inside a <Modify>, it MAY register the urn:liberty:dst:multipleModification discovery option keyword.

- If the processing of a <Modification> fails even partly due to some reason, depending on the service and/or a WSP either the processing of the whole <Modify> MUST fail or a WSP MUST try to achieve partial success. The top level status code Failed or Partial MUST be used to indicate the failure (complete or partial) and a more detailed second level status code SHOULD be used to indicate the reason for failing to completely process the failed <Modify> element. Furthermore, the ref attribute of the <Status> element SHOULD carry the value of the itemID of the failed <Modification> element and in partial success cases it MUST carry this value. The modifications made based on already processed <Modification> elements of the <Modify> MUST be rolled back in case of a complete failure. A WSP MUST NOT support multiple <Modification> elements inside one <Modify>, if it cannot roll back and partial failure is not allowed.

- When multiple <Modification> elements inside one <Modify> element are supported and partial success is allowed, a WSC MUST use the itemID attribute in each <Modification> element so that a WSP can identify the failed parts, when it is returning status information for a partial success.

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

The following rules specify in more detail how modification works:

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

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

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

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

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

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

- When any of the ACC, modifier, ACCTime or modificationTime attributes is used in a resource, the WSP hosting the data service MUST keep their values up to date. When data is modified, the modifier MUST contain the ProviderID of the modifier or have no value, and the modificationTime MUST define the time of the modification or have no value. The ACC MUST define the attribute collection context of the current value of a data element or have no value and the ACCTime MUST define the time, when the current value of the ACC was defined or have no value.

- If the <NewData> contains modifier, modificationTime or ACCTime attributes for any data element, the WSP MUST ignore these and update the values based on other information than those attributes inside the <NewData> provided by the WSC. If the ACC attribute is included for any data element, the WSP MAY accept it, depending on how much it trusts the requesting service provider. The WSP MAY also accept the id attribute provided inside the <NewData> and some services MAY require that the id attribute MUST be provided by the requesting service provider.

- The id attribute MUST NOT be used as a global unique identifier. The value MUST be chosen so that it works only as unique identifier inside the conceptual XML document, and the value of the id attribute SHOULD be kept the same even if the element is otherwise modified. A WSP MAY not even allow changing the value of the id attribute or any other attribute used to distinguish elements with the same name from each other.

- When data is modified based on the <Modify> request, the values of the modificationTime attributes written by the WSP hosting the data service MAY be same for all inserted and updated elements, but there is no guarantee that they will be exactly the same. When the modificationTime attribute is used by a data service, the WSP MUST keep it up to date to indicate the time of the latest modification of an element and update it, when ever a modification is done either using the <Modify> request or some other way. When the modificationTime attribute is used in container elements, the time of a modification MUST be propagated to all ancestor elements of the modified element all the way up to the root element.

Accounting for concurrent updates is handled using the notChangedSince attribute inside the <Modification> element.

- When the notChangedSince attribute is present, the modifications specified by the <Modification> element MUST NOT be made, if any part of the data to be modified has changed since the time specified by the notChangedSince attribute.
  The second level status code ModifiedSince MUST be used to indicate that the modification was not done because the data has been modified since the time specified by the notChangedSince attribute. If a WSP does not support processing of this attribute properly, it MUST NOT make any changes and it MUST return the second level status code ChangeHistoryNotSupported. If a WSP supports this notChangedSince attribute, it MUST also support the changedSince attribute of the <QueryItem> element.

A WSC might not be allowed to make certain modifications or any modifications at all.

- When a WSP processes the <Modification>, it MUST check, whether the resource owner (for example, the Principal) has given consent to the requester to modify the data. To be able to check WSC-specific access rights, the WSP MUST authenticate the WSC (see [LibertySecMech] and [LibertyMetadata]). If the consent check fails for any part of the requested data, the WSP MUST NOT make the modifications requested in the <Modification> element, even when such consent is missing only for some subelement or attribute. The WSP MAY try to get consent from the Principal while processing the request perhaps using an interaction service (for more information see [LibertyInteract]). The processing of the <Modification> element MUST fail, if the modification was not allowed. The second level status code ActionNotAuthorized MAY be used, if it is considered that the privacy of the owner of the resource is not compromised. A WSP might check the access rights at a higher level, before getting to DST processing and MAY return an ID-* Fault Message [LibertySOAPBinding] and not process the <Modify> element at all, if the requesting WSC is not allowed to modify the data.

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

- 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.

- 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.

5.4. Examples

Please note that as the modification examples are based on the [LibertyIDPP] all DST elements are also in the namespace defined in [LibertyIDPP].

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

```
<pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:Modification>
      <pp:Select>/pp:PP/pp:AddressCard</pp:Select>
      <pp:NewData>
         <pp:AddressCard id='98123'>
            <pp:AddressType>urn:liberty:pp:addrType:home</pp:AddressType>
            <pp:Address>
               <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way North</pp:PostalAddress>
               <pp:PostalCode>98503-2341</pp:PostalCode>
               <pp:L>Olympia</pp:L>
               <pp:ST>wa</pp:ST>
               <pp:C>us</pp:C>
            </pp:Address>
         </pp:AddressCard>
      </pp:NewData>
   </pp:Modification>
</pp:Modify>
```

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

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

```
               <pp:PostalCode>98503-2342</pp:PostalCode>
               <pp:L>Olympia</pp:L>
               <pp:ST>wa</pp:ST>
               <pp:C>us</pp:C>
            </pp:Address>
         </pp:AddressCard>
      </pp:NewData>
   </pp:Modification>
</pp:Modify>
```

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

```
<pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:Modification notChangedSince="2003-01-21T12:40:01Z" overrideAllowed="True">
      <pp:Select>/pp:PP/pp:AddressCard[@pp:id='98123']</pp:Select>
      <pp:NewData>
         <pp:AddressCard id="98123">
            <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
            <pp:Address>
               <pp:PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach Way South</pp:PostalAddress>
               <pp:PostalCode>98503-2398</pp:PostalCode>
               <pp:L>Olympia</pp:L>
               <pp:ST>wa</pp:ST>
               <pp:C>us</pp:C>
            </pp:Address>
         </pp:AddressCard>
      </pp:NewData>
   </pp:Modification>
</pp:Modify>
```

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

```
<pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:Modification>
      <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:ad
drType:home']</pp:Select>
      <pp:NewData>
         <pp:AddressCard id="12398">
            <pp:AddressType>urn:liberty:id-sis-pp:addrType:home</pp:AddressType>
            <pp:Address>
               <pp:PostalAddress>1234 Beach Way</pp:PostalAddress>
               <pp:PostalCode>98765-1234</pp:PostalCode>
               <pp:L>Olympia</pp:L>
               <pp:ST>wa</pp:ST>
               <pp:C>us</pp:C>
            </pp:Address>
         </pp:AddressCard>
      </pp:NewData>
   </pp:Modification>
</pp:Modify>
```

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

```
<pp:Modify xmlns:pp="urn:liberty:id-sis-pp:2003-08">
   <pp:ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</pp:ResourceID>
   <pp:Modification overrideAllowed="True">
      <pp:Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addrType:
home']</pp:Select>
   </pp:Modification>
</pp:Modify>
```

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

```
<pp:ModifyResponse xmlns:pp="urn:liberty:id-sis-pp:2003-08"
timeStamp="2003-03-23T03:40:00Z">
   <pp:Status code="OK"/>
</pp:ModifyResponse>
```

6. Subscriptions and Notifications

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

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

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

6.1. Basic elements

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

6.1.1. <Subscription> element

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

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

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

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

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

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

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

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

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

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

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

```
<xs:element name="Subscription">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Select" type="SelectType" minOccurs="0"/>
      <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
      <xs:element name="NotifyTo" type="xs:anyURI"/>
      <xs:element name="NotifyEndedTo" type="xs:anyURI" minOccurs="0"/>
      <xs:element name="Type" type="TypeType" minOccurs="0"/>
      <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="starts" type="xs:dateTime"/>
    <xs:attribute name="expires" type="xs:dateTime"/>
    <xs:attribute name="duration" type="xs:duration"/>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="invokeID" type="IDType"/>
    <xs:attribute name="subscriptionID" type="IDType"/>
    <xs:attribute name="includeData" default="Yes">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Yes"/>
          <xs:enumeration value="No"/>
          <xs:enumeration value="YesWithCommonAttributes"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

6.1.2. <Notification> element

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

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

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

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

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

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

```
<xs:element name="Notification">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Data" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:any minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="invokeID" type="IDType"/>
      <xs:attribute name="subscriptionID" type="IDType"/>
      <xs:attribute ref="changeFormat"/>
      <xs:attribute name="expires" type="xs:dateTime"/>
      <xs:attribute name="duration" type="xs:duration"/>
      <xs:attribute name="endReason" type="xs:anyURI"/>
   </xs:complexType>
</xs:element>
```

6.2. Protocol elements

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

6.2.1. Subscribing notifications and modifying subscriptions

6.2.1.1. <Subscribe> element

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

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

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

- For new subscription, out of invokeID and subscriptionID attributes there is only the invokeID attribute. The subscriptionID attribute MUST NOT be used, when a WSC requests for a new subscription. The rest of parameters are used normally.

- For renewing an existing subscription the subscriptionID attribute MUST be used to identify the right subscription and the expires or the duration attribute MUST be used to indicate the requested lifetime of the subscription. The id MAY be used, if needed. As renewing can be combined with other modifications to an existing subscription, other content is also allowed except the invokeID attribute, which MUST NOT be used.

- For canceling an existing subscription, a WSC MUST use the subscriptionID attribute to identify the subscription and no other content is allowed, except the id MAY be used, if needed.

- For modifying an existing subscription, the subscriptionID attribute MUST be used to identify the right subscription and the changed information MUST be provided. If any of the elements, <Select>, <Type>, <Trigger> or <Extension> has changed, the element must be provided with the requested current content. This means that if one of the triggers has changed and two other triggers haven't, they all must be provided as part of the <Trigger> element. In case some of those four elements doesn't have any content anymore, an empty element must be provided to indicate the deletion, e.g. <Trigger/>. If there is no changes e.g. to <Trigger>, the <Trigger> element is not present in the message.

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

```
<xs:element name="Subscribe" type="SubscribeType"/>
<xs:complexType name="SubscribeType">
   <xs:sequence>
      <xs:group ref="ResourceIDGroup" minOccurs="0"/>
      <xs:element ref="Subscription" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemID" type="IDType"/>
   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
</xs:complexType>
```

6.2.1.2. <SubscribeResponse> element

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

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

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

```
<xs:element name="SubscribeResponse" type="SubscribeResponseType"/>
<xs:complexType name="SubscribeResponseType">
   <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   <xs:attribute name="timeStamp" type="xs:dateTime"/>
</xs:complexType>
```

6.2.1.3. Processing rules

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

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

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

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

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

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

For each successful <Subscription> element in the request a WSP MUST return a <Notification> element in the response, even when no data is returned to confirm the expiration time/duration of the subscription.

- A <Notification> element MUST have the subscriptionID of the subscription it is related to. If the <Subscription> element was sent to cancel all existing subscription, the corresponding <Notification> element in the <SubscribeResponse> MUST NOT have the subscriptionID attribute at all.

- The attribute endReason MUST NOT be used in a <Notification> element, when sending it in a <SubscribeResponse> element. If a subscription was not accepted, the reason is indicated with status codes.

A subscription must contain the address to which notifications should be sent and other information needed by a WSP to be able to send the requested notifications.

- If the <NotifyTo> element is missing from a <Subscription> element, the processing of the <Subscription> MUST fail and a second level status code MissingNotifyToElement indicating this SHOULD be used in addition to the top level status code.

- The <NotifyTo> element MUST contain at least the <Endpoint> element, which MUST contain a complete URI. If the <Endpoint> element is missing from a <NotifyTo> element, the processing of the <Subscription> containing this <NotifyTo> element MUST fail and a second level status code MissingEndpointElement indicating this SHOULD be used in addition to the top level status code. Also, if a WSP is able to verify the validity of the address provided in the <Endpoint> element and finds out that the address is not valid, the processing of the <Subscription> MUST fail and a second level status code InvalidEndpoint indicating this SHOULD be used in addition to the top level status code.

- A WSC SHOULD also provide one or more <SecurityMechID> elements to indicate, which security mechanisms it supports. The <SecurityMechID> elements MUST be specified according to [LibertySecMech] and the mechanisms SHOULD be listed in the order of preference by the WSC. If a WSC does not provide <SecurityMechID> element, the default is urn:liberty:security:2003-08:null:null, which MUST be avoided, when releasing privacy sensitive data (the case almost every time with identity services). A WSP MUST refuse to use mechanisms, which it considers to be not good enough, also Principal owning the resource may have set some security requirements, which a WSP MUST follow. If a WSP wants to get the <SecurityMechID> element, but it is not included, the processing of the <Subscription> element MUST fail and a second level status code MissingSecurityMechIDElement indicating this SHOULD be used in addition to the top level status code. If a WSP does not support or accept any of the proposed security mechanisms, it MUST NOT accept the subscription and so the processing of the <Subscription> element MUST fail and a second level status code SecurityMechIDNotAccepted indicating this SHOULD be used in addition to the top level status code. Some of the security mechanisms require also credentials and a WSC MUST provide those also. If credentials are needed, but they are not provided, the processing of the <Subscription> element MUST fail and a second level status code MissingCredentials indicating this SHOULD be used in addition to the top level status code.

- The information provided in a <NotifyTo> element is also used, when sending end notifications unless separate <NotifyEndedTo> element is provided. The information in a provided <NotifyEndedTo> element replaces the information in a <NotifyTo> element. Please note that a <NotifyEndedTo> element might contain only a different endpoint for end notifications, in which case the same security mechanisms and credentials provided in a <NotifyTo> element are used. The processing rules for the subelements of a <NotifyEndedTo> element are the same as for subelements of a <NotifyTo> element except the <Endpoint> element is not mandatory as the value provided in a <NotifyTo> element may be used (see previous paragraph).

- Please note that if needed credentials expire earlier than a subscription is suppose to expire and a WSC does not provide new credentials before they expire, the subscription will expire as a WSP is not able to send the notifications anymore.

There might be different type of notifications and different triggers causing those. A WSP may not support all different type of features available.

- A WSP MUST follow the processing rules defined in the specification for the service a WSP hosting for the elements `<Type>` and `<Trigger>`. If the use of these elements is not specified for the service, but either of both of them are included in a `<Subscription>` element in a `<Subscribe>` request, the processing of the `<Subscription>` MUST fail and a second level status code indicating this SHOULD be used, either `TypeNotSupported` or `TriggerNotSupported`.

- If a WSP does not support the `<Type>` of the notification a WSC requests, the processing of the `<Subscription>` MUST fail and a second level status code `TypeNotSupported` indicating this SHOULD be used in addition to the top level status code. Similarly if a WSP does not support the `<Trigger>` a WSC requests, the processing of the `<Subscription>` MUST fail and a second level status code `TriggerNotSupported` indicating this SHOULD be used in addition to the top level status code.

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

- If a `<Subscription>` element contains a `starts` attribute, that subscription, if accepted, MUST be valid after the time defined by this `starts` attribute. If there is no `starts` attribute used, then that subscription, if accepted by a WSP, MUST be valid immediately after processing the request. Also, if the time specified by the `starts` attribute is in the past, then that subscription, if accepted by a WSP, MUST be valid immediately after processing the request.

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

- A WSP MAY change the time when a subscription expires from the expiration time requested by a WSC with the `expires` or `duration` attribute. A WSP MAY shorten the expiration time, but it MUST NOT make the expiration time longer. If no `expires` or `duration` attribute is not included in a `<Subscription>` element in a `<Subscribe]>` request from a WSC, a WSP MUST specify the expiration time for the subscription, if expiration times are required either by the service specification or the WSP. A WSP MUST use the same attribute a WSC used in a request to specify the expiration time (`expires` or `duration`), unless otherwise specified in a service specification. A service specification MAY define that only either of those may be used.

- If a WSC wants to renew an existing subscription, which is about to end, it MUST modify that subscription and give a new value for the `expires` or `duration` attribute of that subscription. A WSP MAY modify the new value in the same way as it MAY modify the proposed value for a new subscription.

- When expiration times are used, a WSP MUST include the `expires` or `duration` attribute with the current value in each `<Notification>` element included in a `<SubscribeResponse>` element.

- There are two special cases available using subscriptions expirations. When the `starts` and the `expires` attributes have exactly same values, the meaning is that a notification MUST be sent exactly at that time. When the `duration` attribute is set to zero (e.g. "`P0D`"), the subscription expires immediately after first notification. Note: when a WSC wants to create subscription, which is valid only for a very short time it MUST explicitly specify a very short duration and not try to use a value equally to zero.

A WSC may want to get the current values a the data when subscribing to notifications.

- The default value of the `returnCurrentValues` attribute is `Yes`, so by default a WSP MUST include the current values in the response. If a WSC sets the value of the `returnCurrentValues` to `No`, a WSP MUST NOT return the current values in the response. Also if the `includeData` in the `<Subscription>` element is set to `No`, the data for that subscription MUST NOT be included in the response even, if the `returnCurrentValues` attribute has the value `Yes`.

- The current values of the data addressed by the `<Select>` element of a `<Subscription>` element are included in the `<Data>` element of the corresponding `<Notification>` element. The data inside the `<Data>` element MUST be selected in the same way as in a normal query (see Section 4). If the `includeData` in the `<Subscription>` has the value `YesWithCommonAttributes`, the existing common attributes MUST be returned with the data.

- A WSP MUST NOT return the `<Data>` element, when a WSC is modifying, renewing or canceling existing subscription unless a WSC is modifying the `<Select>` element of a subscription.

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

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

- Note that there can be consent for subscribing to some data element, but not its attributes. A Principal might not want to release the `modifier` attribute, if she does not want to reveal information about which services she uses. If a WSC is not allowed to get all the data, but some data it wants, a WSP SHOULD accept the subscription, but it MAY also reject it. If a subscription is accepted, the data for which there is no consent from the Principal MUST be handled as if there was no data, i.e. that data MUST NOT be returned in the response message, even if the current values should be returned. Also that data MUST NOT be included in the notification messages sent later on.

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

6.2.2. Querying existing subscriptions

6.2.2.1. <QuerySubscriptions> element

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

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

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

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

6.2.2.2. <Subscriptions> element

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

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

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

```
   <xs:element name="Subscriptions" type="SubscriptionsType"/>
   <xs:complexType name="SubscriptionsType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   </xs:complexType>
```

6.2.2.3. Processing rules

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

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

6.2.3. Sending notifications

6.2.3.1. <Notify> element

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

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

```
<xs:element name="Notify" type="NotifyEndedType"/>
<xs:complexType name="NotifyEndedType">
   <xs:sequence>
      <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemID" type="IDType"/>
   <xs:attribute name="timeStamp" type="xs:dateTime"/>
</xs:complexType>
```

6.2.3.2. <NotifyResponse> element

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

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

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

```
<xs:element name="NotifyResponse" type="NotifyEndedResponseType"/>
<xs:complexType name="NotifyEndedResponseType">
   <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
</xs:complexType>
```

6.2.3.3. Processing rules

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

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

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

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

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

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

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

- For change notification a changeFormat attribute MUST be added for a <Data> element to indicate the format used to shown the changes, if a service specification has not mandated only one specific format to be used for this.

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

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

6.2.4. Notifying the end of a subscription

6.2.4.1. <Ended> element

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

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

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

6.2.4.2. <EndedResponse> element

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

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

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

6.2.4.3. Processing rules

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

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

- A <Notification> element inside an end notification message MUST have the attributes endReason and subscriptionID and it MAY also have the id attribute, if needed, but it MUST NOT have any other content. If it has, the receiving WSC SHOULD ignore the other content.

- The value of an endReason attribute SHOULD be either urn:liberty:dst:endreason:unspecified, urn:liberty:dst:endreason:expired, urn:liberty:dst:endreason:credentialsexpired, urn:liberty:dst:endreason:wscnotacknowledging or urn:liberty:dst:endreason:resourcedeleted. It MAY also have some service or implementation specific value. A WSP MUST be careful not to use any values, which might compromise the privacy of a Principal.

- A <Notification> element inside an end notification message MUST have a subscriptionID attribute to identify the subscription which has ended. If the subscriptionID attribute is missing, the processing of that <Notification> element MUST fail and a second level status code MissingSubscriptionID SHOULD be used in addition to a top level status code. If a WSC does not recognize the value of a subscriptionID attribute, the processing of that <Notification> element MUST fail and a second level status code InvalidSubscriptionID SHOULD be used in addition to a top level status code.

- A WSP SHOULD resend an end notification for which it does not get an acknowledgement in reasonable time, if acknowledgements are used.

6.3. Examples

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

7. The Schema for the DST Protocols

```
        <?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:disco="urn:liberty:disco:2004-12"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   xmlns:sb20="urn:liberty:sb:2004-12">

   <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
   <xs:import namespace="urn:liberty:disco:2004-12" schemaLocation="liberty-idwsf-disco-svc
-v2.0.xsd"/>
   <xs:import namespace="urn:liberty:sb:20
04-12" schemaLocation="liberty-idwsf-soap-binding-v2.0.xsd"/>

   <xs:annotation>
      <xs:documentation>
The source code in this XSD file was excerpted verbatim from:

Liberty ID-WSF Data Services Template Specification
Version 2.0-06 Draft
22 November 2004

Copyright (c) 2004 Liberty Alliance participants, see
http://www.projectliberty.org/specs/idwsf_copyrights.html

NOTE: This schema must be used within the context of another schema -
It is not intended to validate by itself.

The scheme which includes this must provide definitions for:
TypeType
SelectType
TriggerType

      </xs:documentation>
   </xs:annotation>
   <xs:element name="ResourceID" type="disco:ResourceIDType"/>
   <xs:element name="EncryptedResourceID" type="disco:EncryptedResourceIDType"/>
   <xs:group name="ResourceIDGroup">
      <xs:choice>
         <xs:element ref="ResourceID"/>
         <xs:element ref="EncryptedResourceID"/>
      </xs:choice>
   </xs:group>
   <xs:element name="ChangeFormat">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ChangedElements"/>
            <xs:enumeration value="CurrentElements"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:element>
   <xs:attribute name="changeFormat">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ChangedElements"/>
            <xs:enumeration value="CurrentElements"/>
            <xs:enumeration value="All"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <!-- Querying Data -->
   <xs:element name="Query" type="QueryType"/>
   <xs:complexType name="QueryType">
      <xs:sequence>
```

```
        <xs:group ref="ResourceIDGroup" minOccurs="0"/>
        <xs:element name="QueryItem" minOccurs="0" maxOccurs="unbounded">
           <xs:complexType>
              <xs:sequence>
                 <xs:annotation>
                    <xs:documentation>
                       NOTE: The below two types (SelectType and SortType) must
                       be defined by the schema that includes this one.
                    </xs:documentation>
                 </xs:annotation>
                 <xs:element name="Select" type="SelectType" minOccurs="0"/>
                 <xs:element name="Sort" type="SortType" minOccurs="0"/>
                 <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
              </xs:sequence>
              <xs:attribute name="id" type="xs:ID"/>
              <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="0"/>
              <xs:attribute name="itemID" type="IDType"/>
              <xs:attribute name="changedSince" type="xs:dateTime"/>
              <xs:attribute name="count" type="xs:nonNegativeInteger"/>
              <xs:attribute name="offset" type="xs:nonNegativeInteger" default="0"/>
              <xs:attribute name="setID" type="IDType"/>
              <xs:attribute name="setReq">
                 <xs:simpleType>
                    <xs:restriction base="xs:string">
                       <xs:enumeration value="Static"/>
                       <xs:enumeration value="DeleteSet"/>
                    </xs:restriction>
                 </xs:simpleType>
              </xs:attribute>
           </xs:complexType>
        </xs:element>
        <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:ID"/>
     <xs:attribute name="itemID" type="IDType"/>
  </xs:complexType>
  <xs:element name="QueryResponse" type="QueryResponseType"/>
  <xs:complexType name="QueryResponseType">
     <xs:sequence>
        <xs:element ref="Status"/>
        <xs:element name="Data" minOccurs="0" maxOccurs="unbounded">
           <xs:complexType>
              <xs:sequence>
                 <xs:any minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="id" type="xs:ID"/>
              <xs:attribute name="itemIDRef" type="IDReferenceType"/>
              <xs:attribute name="notSorted">
                 <xs:simpleType>
                    <xs:restriction base="xs:string">
                       <xs:enumeration value="Now"/>
                       <xs:enumeration value="Never"/>
                    </xs:restriction>
                 </xs:simpleType>
              </xs:attribute>
              <xs:attribute ref="changeFormat"/>
              <xs:attribute name="remaining" type="xs:integer"/>
              <xs:attribute name="nextOffset" type="xs:nonNegativeInteger" default="0"/>
              <xs:attribute name="setID" type="IDType"/>
           </xs:complexType>
        </xs:element>
        <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:ID"/>
     <xs:attribute name="itemIDRef" type="IDReferenceType"/>
     <xs:attribute name="timeStamp" type="xs:dateTime"/>
  </xs:complexType>
```

```
<!-- Modifying Data -->
<xs:element name="Modify" type="ModifyType"/>
<xs:complexType name="ModifyType">
   <xs:sequence>
      <xs:group ref="ResourceIDGroup" minOccurs="0"/>
      <xs:element name="Modification" maxOccurs="unbounded">
         <xs:complexType>
            <xs:sequence>
               <xs:annotation>
                  <xs:documentation>
                     NOTE: The below SelectType must be defined by
                     the schema that includes this one.
                  </xs:documentation>
               </xs:annotation>
               <xs:element name="Select" type="SelectType" minOccurs="0"/>
               <xs:element name="NewData" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:any minOccurs="0" maxOccurs="unbounded"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID"/>
            <xs:attribute name="itemID" type="IDType"/>
            <xs:attribute name="notChangedSince" type="xs:dateTime"/>
            <xs:attribute name="overrideAllowed" type="xs:boolean" default="0"/>
         </xs:complexType>
      </xs:element>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemID" type="IDType"/>
</xs:complexType>
<xs:element name="ModifyResponse" type="ResponseType"/>
<xs:complexType name="ResponseType">
   <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   <xs:attribute name="timeStamp" type="xs:dateTime"/>
</xs:complexType>
<!-- Subscribing notifications and modifying, renewing and deleting existing notifications -->
<xs:element name="Subscribe" type="SubscribeType"/>
<xs:complexType name="SubscribeType">
   <xs:sequence>
      <xs:group ref="ResourceIDGroup" minOccurs="0"/>
      <xs:element ref="Subscription" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemID" type="IDType"/>
   <xs:attribute name="returnCurrentValues" type="xs:boolean" default="1"/>
</xs:complexType>
<xs:element name="SubscribeResponse" type="SubscribeResponseType"/>
<xs:complexType name="SubscribeResponseType">
   <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   <xs:attribute name="timeStamp" type="xs:dateTime"/>
</xs:complexType>
```

```
<!-- Query subscriptions -->
<xs:element name="QuerySubscriptions" type="QuerySubscriptionsType"/>
<xs:complexType name="QuerySubscriptionsType">
   <xs:sequence>
      <xs:group ref="ResourceIDGroup" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemID" type="IDType"/>
</xs:complexType>
<xs:element name="Subscriptions" type="SubscriptionsType"/>
<xs:complexType name="SubscriptionsType">
   <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element ref="Subscription" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
</xs:complexType>
<!-- Subscription Element -->
<xs:element name="Subscription">
   <xs:complexType>
      <xs:sequence>
         <xs:annotation>
            <xs:documentation>
               NOTE: The SelectType, TypeType, and TriggerType below
               must be defined by the schema that includes this one.
            </xs:documentation>
         </xs:annotation>
         <xs:element name="Select" type="SelectType" minOccurs="0"/>
         <xs:element ref="ChangeFormat" minOccurs="0" maxOccurs="2"/>
         <xs:element name="NotifyTo" type="sb20:ServiceInstanceUpdateType"/>
         <xs:element name="NotifyEndedTo" type="sb20:ServiceInstanceUpdateType" minOccurs="0"/>
         <xs:element name="Type" type="TypeType" minOccurs="0"/>
         <xs:element name="Trigger" type="TriggerType" minOccurs="0"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="starts" type="xs:dateTime"/>
      <xs:attribute name="expires" type="xs:dateTime"/>
      <xs:attribute name="duration" type="xs:duration"/>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="invokeID" type="IDType"/>
      <xs:attribute name="subscriptionID" type="IDType"/>
      <xs:attribute name="includeData" default="Yes">
         <xs:simpleType>
            <xs:restriction base="xs:string">
               <xs:enumeration value="Yes"/>
               <xs:enumeration value="No"/>
               <xs:enumeration value="YesWithCommonAttributes"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>

<!-- Sending Notifications and Notifying about subscriptions which have ended-->
<xs:element name="Notify" type="NotifyEndedType"/>
<xs:element name="NotifyResponse" type="NotifyEndedResponseType"/>
<xs:element name="Ended" type="NotifyEndedType"/>
<xs:element name="EndedResponse" type="NotifyEndedResponseType"/>
<xs:complexType name="NotifyEndedType">
   <xs:sequence>
      <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID"/>
```

```
      <xs:attribute name="itemID" type="IDType"/>
      <xs:attribute name="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
   <xs:complexType name="NotifyEndedResponseType">
      <xs:sequence>
         <xs:element ref="Status"/>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   </xs:complexType>
   <!-- Notification Element -->
   <xs:element name="Notification">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Data" minOccurs="0">
               <xs:complexType>
                  <xs:sequence>
                     <xs:any minOccurs="0" maxOccurs="unbounded"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="id" type="xs:ID"/>
         <xs:attribute name="invokeID" type="IDType"/>
         <xs:attribute name="subscriptionID" type="IDType" use="required"/>
         <xs:attribute ref="changeFormat"/>
         <xs:attribute name="expires" type="xs:dateTime"/>
         <xs:attribute name="duration" type="xs:duration"/>
         <xs:attribute name="endReason" type="xs:anyURI"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

8. Checklist for Service Specifications

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

For optional features, the language specified by [RFC2119] MUST be used to define whether these features are available for implementations and deployments. For example, specifying that a feature 'MAY' be implemented by 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:
	
	`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`

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:
	
	`<xs:simpleType name="SelectType">`
	`  <xs:restriction base="xs:string"/>`
	`</xs:simpleType>`

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 `id` attribute MUST be used for `<AddressCard>` and `<MsgContact>` elements, `xml:lang` and `script` 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 `<Query>` elements	Are multiple `<Query>` elements supported?
Multiple `<QueryItem>` elements	Are multiple `<QueryItem>` 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 `<Sort>` element, or a reference to the definition. E.g. when sorting is not supported the type definition using Liberty utility schema could be:
	`<xs:complexType name="SortType">`
	`  <xs:complexContent>`
	`  <xs:restriction base="EmptyType"/>`
	`  </xs:complexContent>`
	`</xs:complexType>`

Support `changedSince`	State here whether the `changedSince` attribute is supported. (for example, this service SHOULD support `changedSince`)
Supported formats	If the `changedSince` attribute is supported, there a three different formats available: `ChangedElements`, `CurrentElements` and `All`. 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 `includeCommonAttributes`	State whether the `includeCommonAttributes` 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 <X> and <Y> 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 <X> and MAY support it for all other elements.
<Extension> in <Query>	Is the <Extension> element inside the <Query> 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 <Modify> elements	If modifications are supported, are multiple <Modify> elements supported?
Multiple <Modification> elements	If modifications are supported, are multiple <Modification> elements supported?
Support partial success	If multiple <Modification> elements are supported, is partial success supported or are only atomic modifications allowed?
Support `notChangedSince`	State here whether the `notChangedSince` attribute is supported. (for example, this service SHOULD support `notChangedSince`)
<Extension> in <Modify>	Is the <Extension> element inside the <Modify> 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	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:
	
	`<xs:complexType name="TypeType">`
	`  <xs:complexContent>`
	`  <xs:restriction base="EmptyType"/>`
	`  </xs:complexContent>`
	`</xs:complexType>`
	
	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.

2880 **Table 9. Subscribe Parameters (2/2)**

Parameter	Value

`TriggerType` definition	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:
	
	`<xs:complexType name="TriggerType">`
	`  <xs:complexContent>`
	`  <xs:restriction base="EmptyType"/>`
	`  </xs:complexContent>`
	`</xs:complexType>`
	
	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.

Start of a subscription	Usually a subscription is valid after it has been created, but if supported, a WSC may request that a subscription is valid only after a specific time using the `starts` attribute. It MUST be specified here, is the `starts` attribute supported or not. E.g. The `starts` attribute MUST NOT be used.
Subscription expiration	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.
Use of `expires` and `duration` attributes	Two different attributes, `expires` and `duration`, 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 `expires` and `duration` MAY be used.
Support `expires==starts`	Is it allowed to specify same time both for the `starts` and the `expires` attribute to request one notification message at a specified time. E.g. same value MAY be used both for the `starts` and the `expires` attribute.
Support zero `duration`	Is it allowed to specify the `duration` 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 `duration` attribute.
`<Extension>` in `<Subscribe>`	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.

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.

References

Normative

[LibertyDisco] Beatty, John, Hodges, Jeff, Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-02, Liberty Alliance Project (24 Nov 2004). *http://www.projectliberty.org/specs*

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

[LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, eds. " Liberty ID-WSF SOAP Binding Specification ," Version 2.0-01, Liberty Alliance Project (22 November 2004). *http://www.projectliberty.org/specs*

[LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 2.0-01, Liberty Alliance Project (22 November 2004). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-01, Liberty Alliance Project (22 November 2004). *http://www.projectliberty.org/specs*

[LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.1, Liberty Alliance Project (18 April 2004). *http://www.projectliberty.org/specs*

[LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version 1.3-errata-v1.0, Liberty Alliance Project (12 Aug 2004). *http://www.projectliberty.org/specs*

[LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-1/*

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

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/2000/REC-xml-20001006*

Informative

[LibertyIDPP] Kellomaki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and Processing," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmldsig-core*