Liberty ID-WSF Data Services Template Specification

Version: 1.0

Editors:
Jukka Kainulainen, Nokia
Aravindan Ranganathan, Sun Microsystems, Inc

Contributors:
Rajeev Angal, Sun Microsystems, Inc
Conor Cahill, AOL
Andy Feng, AOL
Gael Gourmelen, France Telecom
Lena Kannappan, France Telecom
Sampo Kellomaki, Symlabs
John Kemp, IEEE-ISTO
Jonathan Sergent, Sun Microsystems, Inc

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) using the Liberty Identity Web Services Framework (ID-WSF). The specification also defines some guidelines and common XML attributes and data types for data services.

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

Notice

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

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. Additionally, some guidelines, common XML attributes and data types are defined for data services.

This specification doesn't give a strict definition as to 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).

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 can also support actions (e.g. making reservations). Also some services might support only querying data without supporting modifications.

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 that the Data Services Template schemas includes Liberty Utility schema and some elements and types are defined in that schema.

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.

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 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 and the data structure. Typically this 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. 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.
- 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 doesn't conflict with the original attribute definition.
- All elements SHOULD be defined as global elements. 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).

- 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

There are two type of XML elements defined in the XML schemas for the services. Some XML elements contain data a data services is expected to support. One type of XML elements are containers, which 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 3.3). 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 can specify separately if one or more of these common attributes are mandatory 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 doesn't

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 SHOULD be propagated up all the way to the root element.

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 doesn't 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.

2.5. Common Data Types

The type definitions provided by XML schema can't 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:md="urn:liberty:metadata:2003-08" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:import namespace="urn:liberty:metadata:2003-08" schemaLocation="liberty-metadata
-v1.0.xsd"/>
   <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.w3.org/2001/x
ml.xsd"/>
   <xs:include schemaLocation="liberty-idwsf-utility-v1.0.xsd"/>
   <xs:annotation>
      <xs:documentation>Liberty Alliance Project ID-WSF Data Services Template Data Types
Schema</xs:documentation>
      <xs:documentation>
The source code in this XSD file was excerpted verbatim from:

Liberty ID-WSF Data Services Template Specification
Version 1.0
12th November 2003

Copyright (c) 2003 Liberty Alliance participants, see
http://www.projectliberty.org/specs/idwsf_copyrights.html

      </xs:documentation>
   </xs:annotation>
   <!-- Common attributes to be used by different services when found useful/needed-->
   <xs:attribute name="id" type="IDType"/>
   <xs:attribute name="modificationTime" type="xs:dateTime"/>
   <xs:attributeGroup name="commonAttributes">
      <xs:attribute ref="id"/>
      <xs:attribute ref="modificationTime"/>
   </xs:attributeGroup>
   <xs:attribute name="ACC" type="xs:anyURI"/>
   <xs:attribute name="ACCTime" type="xs:dateTime"/>
   <xs:attribute name="modifier" type="md:entityIDType"/>
   <xs:attributeGroup name="leafAttributes">
      <xs:attributeGroup ref="commonAttributes"/>
      <xs:attribute ref="ACC"/>
      <xs:attribute ref="ACCTime"/>
      <xs:attribute ref="modifier"/>
   </xs:attributeGroup>
   <xs:attribute name="script" type="xs:anyURI"/>
   <xs:attributeGroup name="localizedLeafAttributes">
      <xs:attributeGroup ref="leafAttributes"/>
      <xs:attribute ref="xml:lang" use="required"/>
      <xs:attribute ref="script"/>
   </xs:attributeGroup>
   <!-- Common data types to be used by different services when found useful/needed-->
   <xs:complexType name="DSTLocalizedString">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attributeGroup ref="localizedLeafAttributes"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="DSTString">
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attributeGroup ref="leafAttributes"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="DSTInteger">
      <xs:simpleContent>
         <xs:extension base="xs:integer">
            <xs:attributeGroup ref="leafAttributes"/>
         </xs:extension>
```

```
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="DSTURI">
      <xs:simpleContent>
         <xs:extension base="xs:anyURI">
            <xs:attributeGroup ref="leafAttributes"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="DSTDate">
      <xs:simpleContent>
         <xs:extension base="xs:date">
            <xs:attributeGroup ref="leafAttributes"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="DSTMonthDay">
      <xs:simpleContent>
         <xs:extension base="xs:gMonthDay">
            <xs:attributeGroup ref="leafAttributes"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:schema>
```

3. Message Interface

This specification defines two protocols, one for querying data and another for modifying data. These protocols both rely on a request/response message-exchange pattern. The messages specified in this document for those protocols are carried in the SOAP body (see [SOAPv1.1]). No additional content is specified for the SOAP header in this document, but implementers of these protocols MUST follow the rules defined in [LibertySOAPBinding] in addition to those defined more generally for SOAP headers [SOAPv1.1] .

The messages for querying and modifying data have common attributes and elements. These common parts are discussed prior to specifying the actual messages.

3.1. Common Parts

3.1.1. Resources

Both protocols, the one for querying and the one for modifying data, have a defined hierarchy for accessing data. In the first level the desired resources are selected. For example, a resource might be the personal profile of a certain person.

Multiple resources can be accessed in a single request, but querying and modifying can't be mixed in one request message. For each resource there is one `<Query>` or `<Modify>` element in the request message. Inside this element there is another element identifying the resource. This identifying element is either the `<ResourceID>` element or the `<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 `<Query>` or `<Modify>` 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.

3.1.2. <Select> element

The second level of resource selection is inside the `<Query>` and `<Modify>` elements. 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. When only a part of a home address is being queried or modified, the `<Select>` element must point only to that part, or 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 `<Query>` or `<Modify>` element as both of those elements can contain multiple `<Select>` elements in their own sub-structure.

The type of `<Select>` is `SelectType`. Although the type is referenced by *this* specification, the type may vary according to the service specification using this schema, and therefore MUST be defined within each service schema.

When the `SelectType` is specified by 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.

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 doesn't 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.

3.1.3. <Status> element

A response message contains one or more `<Status>` elements to indicate whether or not the processing of the request succeeded. The `<Status>` element is included from the Liberty Utility Schema. A `<Status>` element has a `code` attribute, which contains the return status as a QName. The local part of these codes is specified in this document but the actual values MUST appear in the namespace of the service that includes the DST schema for its protocols.

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`
- `InvalidData`
- `InvalidResourceID`
- `InvalidSelect`
- `MissingNewDataElement`
- `MissingResourceIDElement`
- `MissingSelect`
- `ModifiedSince`

- `NoMoreElements`
- `NoMultipleAllowed`
- `NoMultipleResources`
- `OK`
- `TimeOut`
- `UnexpectedError`

The `<Status>` element may contain another `<Status>` element supplying more detailed return status information. The `code` attribute of the top level `<Status>` element MUST contain either the value `OK` or `Failed`. The remainder of the values above are used to indicate more detailed return status.

If the request 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. When the offending element does not have the `itemID` 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` or `itemID` attribute) the reference SHOULD be made to the ancestor element 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 didn't 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.

3.1.4. Linking with ids

Different types of `id` attributes are used to link queries and responses together. Response messages are correlated with requests using `messageId` and `inResponseToMessageId` attributes that are present in the SOAP Header. Services MUST include `messageId` and `inResponseToMessageId` attributes in all request and response messages defined here. Use of these MUST follow the processing rules specified in [LibertySOAPBinding]. Inside messages, `itemID` and `itemIDRef` attributes are used for linking information inside response messages to the details of request messages. Please note that response messages do not contain the `<ResourceID>` or the `<EncryptedResourceID>` element, so they cannot be used for this.

See the definitions and the processing rules of `<Query>` and `<Modify>` elements for more detailed information.

Some elements in both the request and the response 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 the signature.

3.1.5. The timeStamp Attribute

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

3.1.6. The <Extension> Element

All messages have an `<Extension>` element for services which need more parameters. The `<Extension>` element MUST NOT be used in a message, unless its content and related processing rules have been specified for the service.

3.2. Querying Data

Two different kind of queries are supported, one for retrieving current data, and another for requesting only change 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.

3.2.1. 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. There can be multiple <QueryItem> elements in one <Query>.

The only mandatory content the <QueryItem> element must contain is a <Select> element. The <Select> element specifies the data the query should return. When the select points to one or more data elements, then all of these elements and their descendants are returned.

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. Please note that use of this attribute doesn't 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.

The schema for <Query> is as follows:

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

3.2.2. The <QueryResponse> Element

In addition to different ids the <QueryResponse> can contain three different things: requested data elements, 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 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: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>
```

3.2.3. Processing Rules

A request message can contain multiple <Query> elements. The following rules specify how those must be supported and handled:

- A WSP MUST support one <Query> element inside a request message and SHOULD support multiple. If a WSP supports only one <Query> element inside a request message and the message contains multiple <Query> elements, the processing of the whole message MUST fail and a status code indicating 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 query multiple resources in one message. If a WSP supports accessing multiple resources, it MAY register urn:liberty:dst:multipleResources discovery option keyword.

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

- If processing of a <Query> fails for some reason, any other <Query> elements included in the request SHOULD be processed normally, as if the error had not occurred. When processing of a <Query> fails, 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. A successful query MUST be indicated using the top level status code OK.

The WSP must know which resource the 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 <Query>, the processing of the whole <Query> MUST fail and a status code indicating failure MUST be returned in the response, unless the request was sent using the Liberty Reverse HTTP Binding for SOAP ([LibertyPAOS] and the <ResourceID> element would have the value urn:liberty:isf:implied-resource (see Section 3.1.1 [12]). When either the <ResourceID> or the <EncryptedResourceID> element should have been present, a more detailed status code with the value MissingResourceIDElement SHOULD used in addition to the top level status code.

- If the resource identified in the <ResourceID> or <EncryptedResourceID> element doesn't exist, 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 InvalidResourceID SHOULD be used in addition to the top level status code.

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 the <Select> element is missing from the <QueryItem> element, the processing of that <QueryItem> MUST fail and a status code indicating 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.

- If the <Select> element contains an invalid pointer, for example, to data not supported by the WSP, the processing of that <QueryItem> 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. 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.

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

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

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. A Principal might not want to release the modifier attribute, if she doesn't want to reveal information about which services she uses. The data for which there is no consent from the Principal MUST be handled as if there was no data. 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 a <S:Fault> [SOAPv1.1] without processing the <Query> element at all, or returning a <QueryResponse> 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 which the <Select> directly points to, and which have been modified since the time specified in the changedSince attribute. When the WSP is returning only changed information, it MUST return an empty element, if some element has been deleted, to indicate the deletion (<ElementName/>). 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"/>). 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 the <Select> points to 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. 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.

- 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 the WSP doesn't 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 the WSP returns all current values (and no empty elements for the deleted elements), it SHOULD indicate this with the second level status code AllReturned. 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. The WSP MUST use either AllReturned or ChangedSinceReturnsAll as the second level status code, when it returns data, but doesn't process the changedSince attribute properly, i.e. returns only the changes. If the 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 the 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 the 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. The WSP MUST NOT return empty elements for the data for which access rights were changed, 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 is it 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. The WSP MUST NOT return empty elements for the data for which the Principal's policy was changed, 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 query without a changedSince attribute, before making any data with it.

- As mentioned earlier the WSP might in some cases return all the current data the `<Select>` points to, and not just the changes, 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 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 WSP MUST add a `timeStamp` to the `<QueryResponse>`, if the processing of the `<Query>` was successful and the WSP supports the `changedSince` attribute properly (by keeping track of modification times and trying to return only changes). 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 query for which the `timeStamp` was returned. The value of the `timeStamp` attribute MUST also be such that it can be used as a value for `notChangedSince` attribute, when making modifications after the query for which the `timeStamp` was returned and the modifications will not succeed, if there have been any modifications after this query.

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

The 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 data is not returned to the WSC due to this.

- Other error conditions than those listed in this specification may occur. The second level status code `UnexpectedError` SHOULD be used to indicate such errors.

3.2.4. Examples

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

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

This query may generate the following response:

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

```
<QueryResponse timeStamp="2003-02-28T12:10:12Z">
   <Status code="OK"/>
   <Data itemIDRef="home">
      <AddressCard id='9812'>
         <AddressType>urn:liberty:id-sis-pp:addrType:home<AddressType>
               <Address>
                  <C>us</C>
               </Address>
      </AddressCard>
   </Data>
</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:

```
   <QueryResponse  timeStamp="2003-02-28T12:10:12Z">
      <Status code="OK"/>
   </QueryResponse>
```

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

```
                <Query>
                   <ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</ResourceID>
                   <QueryItem includeCommonAttributes="True">
                      <Select>/pp:PP/pp:LegalIdentity/pp:VAT</Select>
                   </QueryItem>
                </Query>
```

This query may generate the following response:

```
                <QueryResponse id="12345" timeStamp="2003-05-28T23:10:12Z">
                   <Status code="OK"/>
                   <Data>
                      <VAT modifier="http://www.accountingservices.com"
                         modificationTime="2003-04-25T15:42:11Z"
                         attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments">
                         <IDValue modifier="http://www.accountingservices.com"
                            modificationTime="2003-04-25T15:42:11Z"
                            attributeCollectionContext="urn:liberty:dst:acc:secondarydocuments
">502677123</IDValue>
                         <IDType modifier="http://www.accountingservices.com"
                            modificationTime="2003-03-12T09:12:09Z"
                            attributeCollectionContext="urn:liberty:dst:acc:secondarydo
cuments">urn:liberty:altIDType:itcif</IDType>
                      </VAT>
                   </Data>
                </QueryResponse>
                <ds:signature>. . .</ds:signature>
```

The following request queries for address information which has been changed since 12:10.12 28 February 2003 UTC:

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

This query can generate following response:

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

```
<QueryResponse timeStamp="2003-05-30T16:10:12Z">
   <Data/>
   <Status code="OK"/>
</QueryResponse>
```

3.3. 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.

3.3.1. <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 only mandatory content the <Modification> element contains is the <Select> element. The <Select> element specifies the data this modification should affect. 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 doesn't 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 hasn't changed since the time specified by the value of the notChangedSince attribute.

In addition to the id attribute, the <Modify> element can have also the itemID attribute. Thisis 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"/>
               <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="notChangedSince" type="xs:dateTime"/>
                        <xs:attribute name="overrideAllowed" type="xs:boolean"
default="False"/>
                     </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>
```

3.3.2. <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>
```

3.3.3. Processing Rules

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

- A WSP MUST support one <Modify> element inside a request message and SHOULD support multiple. If a WSP supports only one <Modify> element inside a request message and the message contains multiple <Modify> elements, the processing of the whole message MUST fail and a status code indicating failure MUST be returned in the response. A more detailed status code with the value NoMultipleResources SHOULD be used in addition to the top level status code to denote that it is not possible to modify multiple resources with one message. If a WSP supports accessing multiple resources, it MAY register the urn:liberty:dst:multipleResources discovery option keyword.

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

- If processing of a <Modify> fails due to some reason, any other <Modify> elements in the message SHOULD be processed normally, if they haven't been processed already. When processing of a <Modify> fails, the top level status code Failed MUST be used to indicate the failure and a more detailed status code SHOULD be used to indicate the reason for failing to completely process the failed <Modify> element. A successful case MUST be indicated using the top level status code OK.

The WSP must know which resource the WSC wants to access to be able to process the query.

- If there is no <ResourceID> or <EncryptedResourceID> element in the <Modify>, the processing of the whole <Modify> 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> element MAY be left out. When either the <ResourceID> or the <EncryptedResourceID>element should have been present, the value MissingResourceIDElement SHOULD be used for the second level status code.

- If the resource identified by the <ResourceID> or <EncryptedResourceID> element doesn't exist, the processing of the whole <Modify> MUST fail and a status code indicating failure MUST be returned in the response. The value InvalidResourceID SHOULD be used for the second level status code.

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, the processing of the whole <Modify> MUST also fail. The top level status code Failed MUST be used to indicate the failure 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. The modifications made based on already processed <Modification> elements of the <Modify> MUST be rolled back. A WSP MUST NOT support multiple <Modification> elements inside one <Modify>, if it cannot roll back.

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:

- If the <Select> element is missing from the <Modification> element, the processing of that <Modification> MUST fail and a status code indicating a failure MUST be returned in the response. The value MissingSelect SHOULD be used for the second level status code.

- If the <Select> element points to an invalid place, i.e. data not supported by the WSP, the processing of that <Modification> MUST fail and the second level status code InvalidSelect SHOULD be returned in addition to the top level status code in the response. Note that a data service can be extensible and it might not be possible to predefine the exact set of allowed values for the <Select>, if the WSP supports extension.

- When adding new data, the <Select> element will point in the conceptual XML document to an element which doesn't 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.

- A WSP may not support modifications at all. In this case, the second level status code ActionNotSupported SHOULD be returned in addition to the top level status code. A WSP MAY also register the urn:liberty:dst:noModify discovery option keyword to indicate that it does not support modifications at all.

- 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 doesn't 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> doesn't yet exist in the conceptual XML document. The top level status code Failed MUST be returned and also the second level status code ExistsAlready SHOULD be used to indicate in details the reason for the failure. 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 top level status code Failed MUST be returned and 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 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 doesn't 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.

There are some additional rules for handling the common attributes in case of modifications.

- The common attributes belonging to the attribute groups `commonAttributes` and `leafAttributes` are mainly supposed to be written by the WSP hosting the data service.
  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.

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.

- The WSP MUST add a `timeStamp` to the `<ModifyResponse>`, if it supports the `notChangedSince` attribute and the processing of the `<Modify>` was successful. The `timeStamp` attribute MUST have a value, which can also be used as a value for `changedSince` attribute, when querying changes made after the modification for which the `timeStamp` was returned. The value of the`timeStamp` attribute MUST be also such that it can be used as a value for `notChangedSince` attribute, when modifying the data just modified any time after the modification for which the `timeStamp` was returned and the modification will not succeed, if there was some other modification made between these two modifications. The time stamp MUST NOT be older than the latest timestamp stored in the `modificationTime` attributes for elements changed during the modification.

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]). A top level status code of `Failed` MUST be returned, if the modification was not allowed. The second level status code `ActionNotAuthorized` MAY also 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 a `<S:Fault>` [SOAPv1.1] 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 return the top level status code `Failed`. 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 return the top level status code `Failed`. The WSP should use a second level status code `DataTooLong` to indicate this specific case.

The WSP may encounter also other problems than errors in the incoming message.

- If 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 that the requested modification was not made for this reason.

- Error conditions other than those listed in this specification may occur. The second level status code `UnexpectedError` SHOULD be used to indicate this.

The WSP may not always return detailed status codes.

- If the more detailed values for status codes mentioned above are not used to indicate a failure, the value `Failed` MUST be used to indicate a failure.

3.3.4. Examples

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

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

```
                <Modify>
                   <ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</ResourceID>
                   <Modification overrideAllowed="True">
                      <Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-pp:addr
Type:home']</Select>
                      <NewData>
                         <AddressCard id='98123'>
                            <AddressType>urn:liberty:id-sis-pp:addrType:home<AddressType>
                                  <Address>
                                     <PostalAddress>c/o Carolyn Lewis$2378 Madrona Beach↩
Way</PostalAddress>
                                     <PostalCode>98503-2342</PostalCode>
                                     <L>Olympia</L>
                                     <ST>wa</ST>
                                     <C>us</C>
                                  </Address>
                         </AddressCard>
                      </NewData>
                   </Modification>
                </Modify>
```

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

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

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

```
                <Modify>
                   <ResourceID>http://profile-provider.com/d8ddw6dd7m28v628</ResourceID>
                   <Modification>
                      <Select>/pp:PP/pp:AddressCard[pp:AddressType='urn:liberty:id-sis-p
p:addrType:home']</Select>
                      <NewData>
```

```
                        <AddressCard id='12398'>
                           <AddressType>urn:liberty:id-sis-pp:addrType:home<AddressType>
                                 <Address>
                                    <PostalAddress>1234 Beach Way$98765-1234</PostalCode>
                                       <L>Olympia</L>
                                       <ST>wa</ST>
                                       <C>us</C>
                                 </Address>
                        </AddressCard>
                     </NewData>
                  </Modification>
               </Modify>
```

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

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

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

```
               <ModifyResponse timeStamp="2003-03-23T03:40:00Z">
                  <Status code="OK"/>
               </ModifyResponse>
```

3.4. The Schema for Protocol for Querying and Modifying Data.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:disco="urn:liberty:disco:2003-08" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include schemaLocation="liberty-idwsf-utility-v1.0.xsd"/>
   <xs:import namespace="urn:liberty:disco:2003-08" schemaLocation="liberty-idwsf-disco-svc-v1.0.xsd"/>

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

Liberty ID-WSF Data Services Template Specification
Version 1.0
12th November 2003

Copyright (c) 2003 Liberty Alliance participants, see
http://www.projectliberty.org/specs/idwsf_copyrights.html

      </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>
   <!-- Querying Data -->
   <xs:element name="Query" type="QueryType"/>
   <xs:complexType name="QueryType">
      <xs:sequence>
         <xs:group ref="ResourceIDGroup" minOccurs="0"/>
         <xs:element name="QueryItem" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Select" type="SelectType"/>
               </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: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: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:element name="Select" type="SelectType"/>
                  <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="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>
</xs:schema>
```

4. 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.

1328 Table 1. Service Parameters

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`

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>`

Query Language	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.
Multiple `<Query>` elements	Are multiple `<Query>` elements supported?
Multiple `<QueryItem>` elements	Are multiple `<QueryItem>` elements supported?
Support modification	Some services or implementations may or may not support modifications. This should be stated here.

1329 Table 2. Service Parameters Continued

Parameter	**Value**
Multiple `<Modify>` elements	If modifications are supported, are multiple `<Modify>` elements supported?
Multiple `<Modification>` elements	If modifications are supported, are multiple `<Modification>` elements supported?
`<Extension>` in `<Query>`	Is the `<Extension>` element inside the `<Query>` element used? If so, for what purpose?
`<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.
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.
Support `changedSince` and `notChangedSince`	State here whether the `changedSince` and the `notChangedSince` attributes are supported. (for example, this service SHOULD support `changedSince`)
Support `includeCommonAttributes`	State whether the `includeCommonAttributes` attribute is supported. (MUST be, or SHOULD be for example)
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.

References

Normative

[LibertyDisco] Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertySOAPBinding] Hodges, Jeff, Aarts, Robert, eds. " Liberty ID-WSF SOAP Binding Specification ," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertyGlossary] Wason, Thomas, eds. "Liberty Technical Glossary," Version 1.2, Liberty Alliance Project (12 November 2003). *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, H.S., Beech, D., Maloney, M., Mendleshon, N., 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). *ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt*

[SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Thatte, Satish, Winer, Dave, eds. World Wide Web Consortium W3C Note (08 May 2000). *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ [http://www.w3.org/TR/2000/NOTE-SOAP-20000508/]*

[XML] Bray, T., Paoli, J., 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] Kellomäki, Sampo, eds. "Liberty Identity Personal Profile Service Specification ," Version 1.0,Liberty Alliance Project(12 November 2003). *http://www.projectliberty/specs*

[XMLDsig] Eastlake, D., Reagle, J., Solo, D., eds. (12 Feb 20002). "XML-Signature Syntax and Processing," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmldsig-core*