ID-WSF Data Service Template

Version: 1.0-16

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

Contributors:
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:

The ID-WSF Data Services Template specification provides protocols for querying and modifying of data attributes when implementing a data service (e.g. Personal Profile service) on using the Liberty Identity Web Services Framework (ID-WSF). The specification defines also some guidelines and common attributes for data services.

Notice

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

Revision History

Revision: 12 **Date:** 05 Mar 2003

Updated as agreed in Houston TEG F2F. More text for describing the conceptual aspect of the XML document added. Separate attribute groups for containers and leaf elements. Status before Data in QueryResponse. Time stamps added. Some processing rules clarified. Multiple modifications in one message not allowed, if roll back is not supported. Some bugs in modify examples fixed.

Revision: 13 **Date:** 11 Mar 2003

Editorial type issues listed at the end of spec in revision 12 are taken care of. Issues 61 and 83 has been fixed. See bugzilla for more details. Issues 57, 59, 60 and 81 are not resolved, 57, 59 and 60 need updated utility schema and 81 needs to be agreed in architecture team.

Revision: 14 **Date:** 28 Mar 2003

Issues 57, 59, 60, 112, 132 and 133 solved and closed in Bugzilla. Improvements done for issues 121 and 128. Partly solved issue 122. Issue 81 is still open, but it should not require any work for DST.

Revision: 15 **Date:** 06 Apr 2003

Removed target namespace from DST (issue 173). Issue 185 filed so that Resource element will be updated in the future, if needed.

Revision: 16 **Date:** 10 Apr 2003

Typos fixed and minor enhancement of the text. Issues 121, 122, 128 and 185 closed. Issue 81 moved to IS as it has nothing to do with DST anymore.

Contents

1. Overview

This specification provides protocols for querying and modifying of data attributes of a Principal from a data service. Also some guidelines and common XML attributes are defined for data services.

This specification doesn't give a strict definition which services are data services and which are not, i.e. for which services this specification is targetted. 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 are not stored by an external entity, the data service knows or generates 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 specified here. They are not restricted to support just querying and modifying data attributes and can support 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 to be used by different data services are defined. After that different ways of accessing the data are defined and the whole XML schema for this Data Services Template are at the end.

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 interoperation 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 definition:

- The prefix `xs:` stands for the W3C XML schema namespace (`http://www.w3.org/2001/XMLSchema`). [SCHEMA1][SCHEMA2]
- The prefix `md:` stands for the Liberty Metadata schema namespace (`http://www.projectliberty.org/schema/metadata#`). [META]

The following namespaces are used in examples:

- The prefix `idpp:` stands for the ID-Personal Profile namespace (`urn:liberty:idpp:1.0`). [IDPP]. Note that the Data Services Template doesn't have any namespace. It is a utility schema to be included by the data services. The Data Services Template schema will appear in the namespace of the data services. This specification uses in examples the ID-Personal Profile service and the `idpp:` is the default namespace used in examples. Note that the Data Services Template schema includes Liberty Utility schema [LU] and some elements and types are defined in that schema.

- The prefix `saml:` stands for the Security Assertion Markup Language (SAML) namespace (`urn:oasis:names:tc:SAML:1.0:assertion`). [SAML]

- 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 instance

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

2. Data Model

Each data service must be specified using the XML schema. The XML schema of a service specifies the data the service can host. So typically the XML schema for a service defines the data and the data structure. This structure is defined for communication between different parties. The data can be stored using this structure and the data can be read from this structure. Individual braches of the structure can be accessed separately. The data can be stored in what ever way using e.g. different data base technologies but that is not visible to other parties. This means also that the XML document defined by the schema is a conceptual XML document. Depending on the backend implementation, there is no XML document. Only fragments of that conceptual document are transfered between different parties, but the XML document is not stored anywhere as an XML document.

The schemas for different services may have common charasteristics. This chapter describes the commonalities specified by the Data Services Template and also gives some 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 subelements.
- 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. Time values and syncronization

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 [SCHEMA2]. 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.

Different parties SHOULD NOT count on that other parties have their clocks synchronized closer than one minute.

2.3. 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. These XML element are leaf elements, they do not contain other XML elements. The other type of XML elements are containers, which do not have any other data content than other XML elements and possible qualifying XML attributes.

Each XML element, both leaf and container elements, can have service specific XML attributes, but there are also common XML attributes for all data services. These common XML attributes are technical attributes, which are usually created by the Web Service Provider hosting a data service (for more details, see Modifying Data chapter). 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. The common attributes are defined as attribute groups. There are two attribute groups, one common and one for the leaf elements. The attribute group for the leaf elements includes the common attribute group.

Common attributes:

- `id` [Optional]
  The `id` is a unique identifier within a document. It can be used to refer to exactly same element later on especially when there can be many 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 that is needed, this `id` attribute must be used. This `id` attribute is only meant for distinguishing XML elements within the same conceptual XML document, so it MUST NOT be used for globally unique identifiers, because that would create privacy problems. The value of the `id` attribute SHOULD stay the same when the content of the element is modified and so the same value of the `id` attribute can be used when querying the same elements at different times.

- `modificationTime` [Optional]
  The `modificationTime` defines the latest time when the element has been modified. Modification means changing the value of the element or any subelement it has. So the time of the modification is propagated up all the way to the root element.

Attributes for leaf elements only:

- `modifier` [Optional]
  The `modifier` is the ProviderID [META] of the service provider which has modified the data element latest.

- `ACC` [Optional]
  The ACC stands for Attribute Collection Context which describes the context (or mechanism) used in collecting the data. This might give useful information to the service provider asking for the data, e.g. has any validation 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 predefined values for the `ACC`:

- `urn:liberty:dst:acc:unknown`
  This mean 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 equals to no `ACC` attribute at all.

- `urn:liberty:dst:acc:incentive`
  There has been some incentive for user to supply correct input (e.g. present sent to the user).

- `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 send to mobile phone number and the message contained a WAP URL to be clicked)

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

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

Other values are also allowed for `ACC`, but this specification defines only the values listed above.
D1When 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 the value for the `ACC` attribute has been given. Note that this can be different that the `modificationTime`. The `ACC` contains information e.g. related to the validation of the entry. The validation might happen later than the value for the actual data entry has been given. Also the entry can be revalidated later on.

The schema for common attributes

```
<xs:attributeGroup name="commonAttributes">
   <xs:attribute name="id" type="IDType"/>
   <xs:attribute name="modificationTime" type="xs:dateTime"/>
</xs:attributeGroup>
<xs:attributeGroup name="leafAttributes">
   <xs:attributeGroup ref="commonAttributes"/>
   <xs:attribute name="modifier" type="md:entityIDType"/>
   <xs:attribute name="ACC" type="xs:anyURI"/>
   <xs:attribute name="ACCTime" type="xs:dateTime"/>
</xs:attributeGroup>
```

3. Message Interface

This specification defines two protocols, one for querying data and another for modifying data. These both protocols rely on the request/response message-exchange model; for a `<Query>` there is always a `<QueryResponse>` (for `<Modify>` a `<ModifyResponse>`). The messages specified in this document for those protocols are carried in the SOAP body (see [SOAP]). No additional content is specified for the SOAP header in this document, but implementors of these protocols MUST follow the rules defined in [SOAPBIND] in addition to those defined more generally for SOAP headers [SOAP].

The request messages have two parts, which are common for both. The request message must state the `<Resource>` it wants to access. The `<Resource>` can be e.g. a Personal Profile of a certain person. For more information about Resources see [DISCO]. The request message must also specify more detailed what it wants to access inside the specified `<Resource>`. This is specified in `<Select>` elements. E.g. when the `<Resource>` is a Personal Profile of a person, the `<Select>` can point to home address. The type of `<Select>` is `xs:anyType` and it is specified in more details in specifications for each service type. Typically the `<Select>` points to some place(s) in the conceptual XML document. E.g. the type and allowed values for the `<Select>` element for the ID-Personal Profile service are specified in [IDPP].

Both `<Query>` and `<Modify>` support multiple operations in one message. The operations must be of same type (either queries or modifications) and be related to the same `<Resource>`. It is not possible to have both queries and modifications in the same message.

A response message contains a `<Status>` element, which indicates, how the processing of the request succeeded. The `<Status>` element is included from the LIberty Utility Schema [LU]. The codes for return statuses are specified in this document. Note that the response messages do not contain the `<Resource>`. 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 [SOAPBIND] .

Some elements in both the request and the response messages can have `id` attributes of type`xs:ID`. These `id` attributes are needed, when some part of the message points to those element. E.g. if usage directives are used, then the usage directive element must point to the correct element. Also some parts of the messages can be signed and the `id` attribute is needed to be able to point to the right parts to indicate, what has been signed.

A response message can also have a time stamp. This time stamp is provided so that the requesting party can later check, has there been any changes since last time.

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

3.1. Querying Data

Two different kind of queries are supported, current data can be queried or only the changed data. These two different kind of queries can be mixed in the same message. The response can contain the data with or without the common technical attributes depending on the request.

3.1.1. `<Query>` element

The `<Query>` element has two subelements. The `<Resource>` element specifies, which resources this query is about. The `<QueryItem>` element specifies, what data the requestor wants from the resource. There can be multiple `<QueryItem>` elements in one `<Query>`.

The only mandatory content the `<QueryItem>` element must have is the `<Select>` element. The `<Select>` element specifies the data the query should return. If the select points to data elements which have no values, then no data

is returned. When the select points to one or more data elements, then all those elements and their descendants are returned.

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

- `includeCommonAttributes` [Optional]
  The `includeCommonAttributes` specifies, what kind of a 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 distiguishing similar elements from each other by the service, it MUST be returned always, even if the `includeCommonAttributes` is false

- `changedSince` [Optional]
  The changeSince attribute should be used, when the requestor wants to get only the data which has changed since the time specified by this attribute. Note that more than only those changed leaf elements might be returned to better indicate, what has changed, e.g. it is not very meaningful to know that a ZIP code has changed, if you do not know for which address that changed ZIP code belongs to. Please note also that this changedSince 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 `itemID` attribute. This `itemID` attribute is needed, when the `<Query>` element has multiple `<QueryItem>` elements. The response message can refer to `itemID` attributes of the `<QueryItem>` elements.

The schema for `<Query>`:

```
<xs:element name="Query" type="QueryType"/>
<xs:complexType name="QueryType">
   <xs:sequence>
      <xs:element ref="Resource"/>
      <xs:element name="QueryItem" maxOccurs="unbounded">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="Select" type="xs:anyType"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID"/>
            <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="false"/>
            <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:complexType>
```

3.1.2. `<QueryResponse>` element

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 corresponding `<QueryItem>` elements with the `itemIDRef` attributes.

The schema for `<QueryResponse>`:

```
<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="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
```

3.1.3. Processing Rules

- If the `<Resource>` element is missing from the `<Query>`, the processing of the whole `<Query>` MUST fail and a status code indicating failure MUST be returned in the response. The status code SHOULD have value`MissingResource`.

- If the resource specified in the `<Resource>` element doesn't exist, the processing of the whole `<Query>` MUST fail and a status code indicating failure MUST be returned in the response. The status code SHOULD have value`NoResource`.

- The `<Query>` can contains multiple `<QueryItem>` elements. 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. The status code SHOULD have value`NoMultipleAllowed`.

- If the `<Query>` contains multiple `<QueryItem>` elements, the WSC MAY add `itemID` attributes for 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 due to some reason, the possible other `<QueryItem>` SHOULD NOT be processed, if they haven't been so far. The data for the processed `<QueryItem>` elements SHOULD be returned return in the response message and the status code MUST indicate the reason for failing to completely process the whole `<Query>`. This status SHOULD indicate the reason for failure with the first failed `<QueryItem>` as the other SHOULD NOT be processed.

- 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. The status code SHOULD have value`MissingSelect`.

- If the `<Select>` element points to an invalid place, i.e. 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. The status code SHOULD have value`InvalidSelect`. Note that a data service can be extendable and it might not be possible to predefine the exact set of allowed values for the `<Select>`, if the WSP supports the extension.

- If 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 for that `<QueryItem>`.

- If the `<QueryItem>` element contains the `changedSince` attribute, the WSP MUST return only those elements which the `<Select>` directly points to and which have any content modified since the time specified in the `changedSince` attribute. Even, if only a single subelement of an element pointed by the `<Select>` has changed, the WSP MUST return the whole element pointed by the `<Select>`. Otherwise it might be difficult to link the changed information to the right context and also this way it is easy to present deleted subelements, when they are not included in the response and all existing ones are. 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/>`) to differenciate from the case, when there are no values or previous values have been deleted. In that case no `<Data>` element is returned as mentioned earlier.

- If the `<QueryItem>` element contains the `changedSince` attribute and the WSP is not keeping track of the modification times, the WSP SHOULD process the `<QueryItem>` element as there is no `changedSince` attribute and it MAY indicate this with status code `changedSinceReturnsAll` in the response. This is not consider as a failure. If the WSP will not process the `<QueryItem>` elements with `changedSince` attribute, it MUST indicate this with status code `changedSinceNotSupported` in the response.

- The WSP MUST add `timeStamp` in the `<QueryResponse>`, if it supports the `changedSince` attribute properly, i.e. it keeps track of modification times and can check, what has really changed since a specified time. The `timeStamp` attribute SHOULD have a value, which can be used as a value for `changedSince` attribute, when querying changes done after the query for which the `timeStamp` was returned.

- 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. If the `id` attribute is used for distiguishing similar elements from each other by the service, it MUST be returned always, even if the `includeCommonAttributes` is false. The `ACC` attributes MAY be left out, if the value is `urn:liberty:dst:acc:unknown`.

- When a WSP processes the `<QueryItem>`, it MUST check, has the entity (e.g. Principal) which can be said to own the resource given the consent to return the requested information back to the requesting party. Also the WSP MUST check that the usage directive given in the request is acceptable based on the usage directives define by the principal or any other entity owning the resource. If either check fails for any part of the requested data, the WSP MUST NOT returned the data, for which there either is no consent or for which the usage directive specified in the request was not acceptable. Note that there can be consent for returning some data element, but not its its attributes. E.g. a Principal might not want to release modifier attribute, if he/she doesn't want to reviel information, which services he/she uses. The data for which there is no consent from the Principal MUST be handled like there was no data.

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

3.1.4. Examples

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

```
<Query>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <QueryItem itemID="name">
      <Select>/IDPP/CommonName</Select>
   </QueryItem>
   <QueryItem itemID="home">
      <Select>/IDPP/IDPPAddressCard[IDPPAddressType="urn:liberty:idpp:addrType:home"]</Select>
```

```
   </QueryItem>
</Query>
```

This query can generate the following response:

```
<QueryResponse>
  <Status code="ok"/>
  <Data itemIDRef="name">
      <CommonName>
         <CN>Zita Lopes</CN>
         <LCN lang="jp" script="kana">LKj343asas</LCN>
         <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">
      <IDPPAddressCard id='9812'>
         <IDPPAddressType>urn:liberty:idpp:addrType:home<IDPPAddressType>
         <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>
      </IDPPAddressCard>
   </Data>
</QueryResponse>
```

If there was no user consent for the <idpp:CommonName> and neither for the whole <idpp:IDPPAddressCard> with idpp:IDPPAddressType='urn:liberty:idpp:addrType:home', just for the country information, then the response is just:

```
<QueryResponse>
   <Status code="ok"/>
   <Data itemIDRef="home">
      <IDPPAddressCard id='9812'>
         <IDPPAddressType>urn:liberty:idpp:addrType:home<IDPPAddressType>
         <Address>
            <C>us</C>
         </Address>
      </IDPPAddressCard>
   </Data>
</QueryResponse>
```

If there was no `<idpp:CommonName>` and no `<idpp:IDPPAddressCard>` with `idpp:IDPPAddressType = 'urn:liberty:idpp:addrType:home'`, then the response is just:

```
<QueryResponse>
   <Status code="ok"/>
</QueryResponse>
```

Following request queries the fiscal identification number of the principal with the common attributes:

```
<Query>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <QueryItem>
      <Select includeCommonAttributes="True">/IDPP/LegalIdentity/VAT</Select>
   </QueryItem>
 </Query>
```

This query can generate following response:

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

Following requests queries for address information, which has been changed since 12:10.12 28 February 2003 UTC:

```
<Query>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <QueryItem>
      <Select changedSince="2003-02-28T12:10:12Z">/IDPP/IDPPAddressCard</Select>
   </QueryItem>
</Query>
```

This query can generate following response:

```
<QueryResponse timeStamp="2003-05-30T16:10:12Z">
   <Status code="ok"/>
   <Data>
      <IDPPAddressCard id='9812'>
         <IDPPAddressType>urn:liberty:idpp:addrType:home</IDPPAddressType>
         <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>
      </IDPPAddressCard>
   </Data>
</QueryResponse>
```

Please note that the whole `<idpp:IDPPAddressCard>` is returned even, if only a typing error in the `<idpp:PostalCode>` has been corrected. If only the changed `<idpp:PostalCode>` would be returned, there would be no way of knowing to which address it belongs.

Or, if there has been no changes since the specified time, then the response is just:

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

3.2. 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 date service using the provided user interface, but these modifications can also be done by other service providers. The `<Modify>` element supports all these operations for service providers, which want to modify the data store in data services.

3.2.1. `<Modify>` element

The `<Modify>` element has two subelements. The `<Resource>` element specifies, which resource is modified by this request. The `<Modification>` element specifies, what data 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 must have is the `<Select>` element. The `<Select>` element specifies the data this modification should modify. 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`, that means the `<Modification>` is only allowed to add new data, not to remove or replace existing data.

The schema for `<Modify>`

```
   <xs:element name="Modify" type="ModifyType"/>
   <xs:complexType name="ModifyType">
      <xs:sequence>
         <xs:element ref="Resource"/>
         <xs:element name="Modification" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Select" type="xs:anyType"/>
                  <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="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:complexType>
```

3.2.2. `<ModifyResponse>` element

The <Modify>Response</Modify> element contains the <Status> element, which tells, how the requested modification succeeded, and a possile time stamp attribute, which provides time value that can be used later to check has there been any changes since this modification.

The schema for <Modify>Response</Modify>

```
<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="timeStamp" type="xs:dateTime"/>
</xs:complexType>
```

3.2.3. Processing Rules

- If the <Resource> element is missing from the <Modify>, the processing of the whole <Modify> MUST fail and a status code indicating failure MUST be returned in the response. The value MissingResource SHOULD be used for the status code.

- If the resource specified in the <Resource> 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 NoResource SHOULD be used for the status code.

- 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 status code.

- If the processing of a <Modification> fails due to some reason, the processing of the whole <Modify> MUST also fail and then the right status code indicating the reason for the failure MUST be returned in the response. The modifications done based on already processed <Modification> elements MUST be rolled back. A WSP MUST NOT support multiple <Modification> elements inside one <Modify>, if it can't roll back.

- If the <Select> element is missing from the <Modification> element, the processing of that <Modification> MUST fail and a status code indicating failure MUST be returned in the response. The value MissingSelect SHOULD be used for the 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 status code InvalidSelect MUST be returned in the response. Note that a data service can be extendable and it might not be possible to predefine the exact set of allowed values for the <Select>, if the WSP supports the extension.

- When adding new data, the <Select> element points 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 the cases, when the parent element of the new element does not exist either, it 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 the <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 (e.g. all IDPPAddressCards).

- When a WSP processes the `<Modification>`, it MUST check, has the entity (e.g. Principal) which can be said to own the resource given the consent to the requesting party to modify the data. If the check fails for any part of the requested data, the WSP MUST NOT modify the data, for which there is no consent. Note that a WSP may not support modifications at all. Status code `NotAllowedToModify` MAY be returned.

- 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 subelement, 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 lack of those other subelements inside the `<NewData>` means that they should be removed, which is only possible when `overrideAllowed` attribute equals to True.

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

- When the `<Modification>` element tries to extend the service either by pointing to new data type with the `<Select>` element or having new subelements 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 status code `ExtensionNotSupported`.

- The common attributes belonging to the attribute groups `commonAttributes` and `leafAttributes` are mainly suppose 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 WSP. If the `ACC` attribute is included for any data element, the WSP MAY accept it depending e.g. on the agreement it has with the requesting service provider and 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` MUST NOT be used as a global unique identifier. The value MUST be chosen so that works only as unique identifier inside the conceptual XML document and the value of the `id` SHOULD be kept the same although the value of the element is modified.

- When the 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.

- The WSP MUST add `timeStamp` in the `<QueryResponse>`, if it supports the `changedSince` attribute in queries properly, i.e. it keeps track of modification times and can check, what has really changed since a specified time. The `timeStamp` attribute SHOULD have a value, which can be used as a value for `changedSince` attribute, when querying changes done after the modification for which the `timeStamp` was returned.

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

3.2.4. Examples

Following example adds a home address to the personal profile of a Principal. Please note that this request will fail, if there already was a home address in the personal profile.

```
<Modify>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <Modification>
      <Select>/IDPP/IDPPAddressCard[IDPPAddressType='urn:liberty:idpp:addrType:home']</Select>
      <NewData>
         <IDPPAddressCard id='98123'>
            <IDPPAddressType>urn:liberty:idpp:addrType:home<IDPPAddressType>
            <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>
         </IDPPAddressCard>
      </NewData>
   </Modification>
</Modify>
```

Following example replaces 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 out of those addressed should be replaced by this address.

```
<Modify>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <Modification overrideAllowed="True">
      <Select>/IDPP/IDPPAddressCard[IDPPAddressType='urn:liberty:idpp:addrType:home']</Select>
      <NewData>
         <IDPPAddressCard id='98123'>
            <IDPPAddressType>urn:liberty:idpp:addrType:home<IDPPAddressType>
            <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>
         </IDPPAddressCard>
      </NewData>
   </Modification>
</Modify>
```

Following example replaces current address which has `id` with value '98123' with a new home address.

```
<Modify>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <Modification overrideAllowed="True">
      <Select>/IDPP/IDPPAddressCard[id='98123']</Select>
      <NewData>
         <IDPPAddressCard id='98123'>
            <IDPPAddressType>urn:liberty:idpp:addrType:home<IDPPAddressType>
            <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>
         </IDPPAddressCard>
      </NewData>
   </Modification>
</Modify>
```

Following example adds another home address in the personal profile of a Principal, the existing home addresses are not touched. An id is provided for the new address.

```
<Modify>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <Modification>
      <Select>/IDPP/IDPPAddressCard[IDPPAddressType='urn:liberty:idpp:addrType:home']</Select>
      <NewData>
         <IDPPAddressCard id='12398'>
            <IDPPAddressType>urn:liberty:idpp:addrType:home<IDPPAddressType>
            <Address>
               <PostalAddress>1234 Beach Way$98765-1234</PostalCode>
               <L>Olympia</L>
               <ST>wa</ST>
               <C>us</C>
            </Address>
         </IDPPAddressCard>
      </NewData>
   </Modification>
</Modify>
```

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

```
<Modify>
   <Resource>
      <saml:NameIdentifier>d8ddw6dd7m28v628</saml:NameIdentifier>
   </Resource>
   <Modification overrideAllowed="True">
      <Select>/IDPP/IDPPAddressCard[IDPPAddressType='urn:liberty:idpp:addrType:home']</Select>
   </Modification>
</Modify>
```

The response for a valid `<Modify>` is e.g. just:

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

4. Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:md="http://www.projectliberty.org/schema/metadata#"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:import namespace="http://www.projectliberty.org/schema/metadata#" schemaLocation="draft-arch-metadata-schema.xsd"/>
   <xs:include schemaLocation="lib-arch-iwsf-utility.xsd"/>
   <xs:annotation>
      <xs:documentation>Liberty Alliance Project ID-WSF Data Services Template Schema</xs:documentation>
      <xs:documentation>Copyright 2003 Liberty Alliance Project</xs:documentation>
   </xs:annotation>
   <!- Common attributes to be used by different services when found useful/needed->
   <xs:attributeGroup name="commonAttributes">
      <xs:attribute name="id" type="IDType"/>
      <xs:attribute name="modificationTime" type="xs:dateTime"/>
   </xs:attributeGroup>
   <xs:attributeGroup name="leafAttributes">
      <xs:attributeGroup ref="commonAttributes"/>
      <xs:attribute name="modifier" type="md:entityIDType"/>
      <xs:attribute name="ACC" type="xs:anyURI"/>
      <xs:attribute name="ACCTime" type="xs:dateTime"/>
   </xs:attributeGroup>
   <xs:element name="Resource" type="xs:anyURI"/>
   <!- Querying Data ->
   <xs:element name="Query" type="QueryType"/>
   <xs:complexType name="QueryType">
      <xs:sequence>
         <xs:element ref="Resource"/>
         <xs:element name="QueryItem" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Select" type="xs:anyType"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:ID"/>
               <xs:attribute name="includeCommonAttributes" type="xs:boolean" default="false"/>
               <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: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="timeStamp" type="xs:dateTime"/>
   </xs:complexType>
   <!- Modifying Data ->
   <xs:element name="Modify" type="ModifyType"/>
   <xs:complexType name="ModifyType">
      <xs:sequence>
         <xs:element ref="Resource"/>
         <xs:element name="Modification" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Select" type="xs:anyType"/>
                  <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="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: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="timeStamp" type="xs:dateTime"/>
  </xs:complexType>
</xs:schema>
```

Bibliography

Normative

[DISCO] Sergent, Jonathan, eds. (2003). "ID-WSF Discovery Service," . 1.0 , Liberty Alliance Project <http://www.projectliberty.org> [http://www.projectliberty.org]

[META] Davis, Peter, eds. (2003). "Metadata description and discovery protocols ," . 1.0 , Liberty Alliance Project <http://www.projectliberty.org> [http://www.projectliberty.org]

[SOAPBIND] Hodges, Jeff, Aarts, Robert, eds. (2003). "ID-WSF SOAP Binding," . 1.0 , Liberty Alliance Project <http://www.projectliberty.org> [http://www.projectliberty.org]

[LU] , , eds. (Month 2003). "ID-WSF Utility Schema," . 1.0 , Liberty Alliance Project <http://www.projectliberty.org> [http://www.projectliberty.org]

[GLOSSARY] Mauldin, Hank, Wason, Tom, eds. (Month 2003). "Liberty Glossary," . , Liberty Alliance Project <http://www.projectliberty.org> [http://www.projectliberty.org]

[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> [http://www.w3.org/TR/xmlschema-1]

[SCHEMA2] Biron, P.V., Malhotra, A., eds. (May 2002). "XML Schema Part 2: Datatypes, Recommendation. ," . , World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2> [http://www.w3.org/TR/xmlschema-2]

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

[SOAP] Box, D., eds. (May 2000). "Simple Object Access Protocol (SOAP) 1.1 ," . 1.0 , World Wide Web Consortium <http://www.w3.org/TR/SOAP> [http://www.w3.org/TR/SOAP]

Informative

[IDPP] Kellomaki, Sampo, eds. (2003). "ID-Personal Profile ," . 1.0 , Liberty Alliance Project <http://www.projectliberty.org> [http://www.projectliberty.org]

[SAML] Hallam-Baker, P., Maler , E., eds. (05 November 2002). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML), OASIS Standard ," . 1.0 , Organization for the Advancement of Structured Information Standards <http://www.oasis-open.org/committees/security/#documents> [http://www.oasis-open.org/committees/security/#documents]

[XMLDsig] Eastlake , D., Reagle, J., Solo, D., eds. (12 February 2002). "XML-Signature Syntax and Processing, Recommendation. ," . , World Wide Web Consortium <http://www.w3.org/TR/xmldsig-core> [http://www.w3.org/TR/xmldsig-core]