Liberty ID-WSF Discovery Service Specification

Version: 2.0-errata-v1.0

Editors:
Conor Cahill, Intel Corporation
Jeff Hodges, NeuStar, Inc.

Contributors:
Robert Aarts, Nokia Corporation
John Beatty, Sun Microsystems, Inc.
Carolina Canales-Valenzuela, Ericsson
Darryl Champagne, IEEE-ISTO
Gary Ellison, Sun Microsystems, Inc.
Jukka Kainulainen, Nokia Corporation
John Kemp, Nokia Corporation
Rob Lockhart, IEEE-ISTO
Paul Madsen, NTT, formerly Entrust, Inc.
Jonathan Sergent, Sun Microsystems, Inc.
Greg Whitehead, Hewlett-Packard
Emily Xu, Sun Microsystems, Inc.

Abstract:

This specification defines mechanisms for describing and discovering identity web services.

Filename: liberty-idwsf-disco-svc-2.0-errata-v1.0.pdf

Notice

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

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

Copyright © 2007 2FA Technology; Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.; American Express Company; Amsoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Bank of America Corporation; Beta Systems Software AG; British Telecommunications plc; Computer Associates International, Inc.; Credentica; DataPower Technology, Inc.; Deutsche Telekom AG, T-Com; Diamelle Technologies, Inc.; Diversinet Corp.; Drummond Group Inc.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Falkin Systems LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le développement de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens Ltd.; GSA Office of Governmentwide Policy; Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke & Devrient GmbH; Hewlett-Packard Company; Hochhauser & Co., LLC; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega; Kayak Interactive; Livo Technologies; Luminance Consulting Services; MasterCard International; MedCommons Inc.; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; NTT DoCoMo, Inc.; Netegrity, Inc.; Neustar, Inc.; New Zealand Government State Services Commission; Nippon Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation; RSA Security Inc.; Reactivity Inc.; Royal Mail Group plc; SAP AG; Senforce; Sharp Laboratories of America; Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.; Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security; Trusted Network Technologies; UNINETT AS; UTI; VeriSign, Inc.; Vodafone Group Plc.; Wave Systems Corp. All rights reserved.

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

Contents

1. Introduction

This specification defines a framework for describing and discovering web services in general and *identity web services* in particular. The conceptual model and terminology is first provided to set the context for the rest of the specification. Next, the data types for the information maintained by a Discovery Service are specified. Then the Discovery Service itself is specified.

1.1. Conceptual Model and Terminology

An *identity web service* is defined as a type of web service whose operations are indexed by *identity*. Such services maintain information about, or on behalf of, *Principals* — as represented by their *identities* — and/or perform actions on behalf of Principals. They are also sometimes referred to as simply *identity services*.

There are various types of services, each of which is assigned a unique *service type* identifier, encoded as a *URI* (Uniform Resource Identifier). This *service type URI* maps to exactly one version of an *abstract WSDL* definition of a service, which contains the `<wsdl:types>`, `<wsdl:message>`, and `<wsdl:portType>` elements of a WSDL 1.1 description [WSDLv1.1].

An example of a type of identity web service is a Principal's "calendar service," which could be identified by a URI such as *urn:example:services:calendar:2006-12*. Note the use of the year/month in the service type to identify the version of the abstract WSDL.

A *service instance* is a deployed physical instantiation of a particular type of service. A *service provider* may deploy one or more concrete service instances in the act of deploying a service.

A service instance may be described by a *concrete WSDL* document (including at least the `<wsdl:binding>`, `<wsdl:service>`, and `<wsdl:port>` elements) which contains the *protocol endpoint* and additional information necessary for a client to communicate with a particular service instance. An example of such "additional information" is communication security policy information.

A service instance is hosted by some *provider*, identified by a URI. An example of a service instance is a SOAP-over-HTTP endpoint offering a calendar service, being hosted by some provider.

Thus, a service instance exposes a protocol interface to a set of logical resources, nominally indexed by Principal. A *resource* in this specification is either data related to some *Principal*'s *identity* or a service acting on behalf of some Principal. An example of a resource is a calendar containing appointments for a particular Principal. When a client sends a request message to a service instance, information in the message serves to implicitly identify the resource being acted upon. This is accomplished in one of the following fashions:

- Implicitly (e.g., PAOS exchange [LibertyPAOS]).
- Via a `<TargetIdentity>` header block [LibertySOAPBinding].
- Via supplied security token: it is presumed that a resource of the security token subject, i.e., the Principal itself, is to be accessed.
- Via the endpoint. A service may choose to offer different endpoints for every resource. The simplest case of this is to represent the resource as a part of the query string.

  Caution should be exercised when using this unique endpoint solution as the use of unique endpoints for every resource can release enough information to allow collusion across providers as to the identity of a principal (if multiple providers get the same unique endpoint reference for their local principal, they can figure out that the local principal on their respective environment is the same principal).

A resource commonly has access control policies associated with it. These access control policies are typically under the purview of the entity or entities associated with the resource (in common language, the entity or entities could be said to "own", or "manage", the resource). The access control policies associated with a resource must be enforced by the service instance.

The Discovery Service defined here is not intended to be exclusive. Some identity services meeting the conceptual model may be exposed via other discovery mechanisms. For example, [LibertyPAOS] defines an equivalent discovery mechanism.

1.2. Scope

This specification:

- Specifies service instance endpoint description and enumeration via a profile of W3C Web Services Addressing [WSAv1.0].
- Specifies a Discovery Service facilitating discovery and invocation of service instances.
- A SAML (see [SAMLCore2]) `<Attribute>` element defined such that an Endpoint Reference (EPR) for the Discovery Service itself can be conveyed via SAML assertions. This is known as a *Discovery EPR* or *DS EPR* and also colloquially as the *discovery bootstrap*.

1.3. Notation and Conventions

This specification uses schema documents conforming to W3C XML Schema (see [Schema1-2]) and normative text to describe the syntax and semantics of XML-encoded messages.

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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.

1.3.1. XML Namespaces

The following XML namespaces are referred to in this document:

- The prefix *ds:* represents the Discovery Service namespace. This namespace is the default for instance fragments, type names, and element names in this document. In schema listings, and in examples of Discovery Service messages and fragments thereof, this is the default namespace *when* no prefix is shown:

  urn:liberty:disco:2006-08

- The prefix *saml2:* stands for the SAMLv2 assertion namespace [SAMLCore2]:

  urn:oasis:names:tc:SAML:2.0:assertion

- The prefix *samlp2:* stands for the SAMLv2 protocol namespace [SAMLCore2]:

  urn:oasis:names:tc:SAML:2.0:protocol

- The prefix *sb:* stands for the Liberty Soap Bindings namespace [LibertySOAPBinding]:

  urn:liberty:sb:2006-08

- The prefix *sbf:* stands for the Liberty Soap Bindings Framework namespace [LibertySOAPBinding]:

  urn:liberty:sb

- The prefix *sec:* stands for the Liberty Security Mechanisms namespace [LibertySecMech]:

  urn:liberty:security:2006-08

- The prefix *wsa:* stands for the W3C Web Services Addressing (WSA) namespace [WSAv1.0]:

  http://www.w3.org/@@@@/@@/addressing

- The prefix *wsdl:* stands for the primary WSDL v1.1 namespace [WSDLv1.1]:

  http://schemas.xmlsoap.org/wsdl/

- The prefix *wsdlsoap:* stands for the namespace of the WSDL-SOAP binding [WSDLv1.1]:

  http://schemas.xmlsoap.org/wsdl/soap/

- The prefix *xs:* stands for the W3C XML schema namespace [Schema1-2]:

  http://www.w3.org/2001/XMLSchema

- The prefix *xsi:* stands for the W3C XML schema instance namespace:

  http://www.w3.org/2001/XMLSchema-instance

2. Discovery Service Information Model

This section describes the Discovery Service information model. This model comprises the various data types, and thus information, that are maintained and managed by the Discovery Service, as well as the manner and format in which this information is exchanged between the Discovery Service and its clients.

First, there is a brief non-normative overview describing how *service instances* are referenced, as well as the interactions between the Discovery Service and the various other roles donned by system entities in the ID-WSF framework. Next are the normative definitions of the various elements defined in this specification and used in referencing service instances. Lastly is the Discovery Service WSA Profile, which normatively defines WSA EPRs profiled for use in referencing ID-WSF service instances.

2.1. Overview of Discovery Service Information Model

A *service instance* is a web service at a distinct protocol endpoint. Information about service instances needs to be communicated in various contexts. This specification defines a profile of WSA Endpoint References (EPRs) [WSAv1.0][WSAv1.0-SOAP] such that they can be used to convey service instance information needed by entities wishing to communicate with said service instances. Such "profiled EPRs" are termed "ID-WSF EPRs" in the remainder of this specification.

The general model for ID-WSF system entity interactions from a Principal's perspective is as follows:

- A Principal wielding some *user agent* interacts with some *service provider* and is authenticated in some Liberty-compliant fashion, such that the service provider obtains possession of a *discovery bootstrap* assertion for the Principal. This assertion contains a pointer to the Principal's Discovery Service instance in the form of an ID-WSF EPR.

- Now, the service provider, acting as a *web service consumer* (WSC) and using the ID-WSF EPR obtained above, queries the Principal's Discovery Service for a pointer to some other desired service of the Principal—e.g., the Principal's Profile Service or Calendar Service.

- The Discovery Service returns one or more ID-WSF EPRs to the querying WSC, pointing to the Principal's service instance(s), of the requested type, if any.

- The WSC now employs the returned ID-WSF EPR(s) to interact with the identified service instance(s), which themselves will be acting in the role of a *web service provider*. The WSC returns results as appropriate to the Principal's user agent.

  There are various permutations of this general interaction model. For example, the Principal's user agent may itself act in the role of a WSC. Or, a Principal may not be actively involved in a given interaction—a WSC is simply interacting with a WSP on a Principal's behalf. For example, it may be renewing some contract, such as a magazine subscription.

In order to enable the above Principal's-perspective model, there is a parallel model from the *web service provider's* (WSP) perspective, which is as follows:

- A service instance(s), acting as a WSP, is deployed at some addressable endpoint(s). In this example, the WSP is providing some service(s) on behalf of one or more Principals, e.g., a profile or calendar service.

- The WSP registers itself with the Discovery Service by inserting Service Metadata into the DS using the Service Metadata maintenance operations (defined later in this specification). The Service Metadata describes the WSP's service instance(s) such that the Discovery Service has the necessary information to mint ID-WSF EPRs for a WSC to invoke that WSP.

- The Service Metadata, using appropriate Discovery Service protocol operations (defined later in this specification), is then "associated" with a principal'.

- The above Principal's-perspective model is now enabled.

  There are various permutations of this general WSP-perspective service instance registration model. For example, the same administrative entity may be deploying the both the Discovery Service and the other services and so may employ alternative means, e.g., bulk configuration, to effect service metadata registration with their Discovery Service.

2.2. Versioning in ID-WSF

Versioning applies to both the communications framework and the service itself within Liberty. The Discovery Service is at the center of versioning in Liberty because it is the entity that matches the version capabilities of the WSC to that of the WSP.

The specific areas of versioning include:

- **Service Versioning** — the version of the Service APIs that are available from a service instance.

  The service version is implicitly part of the `<ServiceType>` URI which identifies a specific version of a specific logical service (e.g., a Profile service or a Discovery Service). For example, the service type URI: `urn:liberty:disco:2006-08` represents a specific version of the logical service "Discovery Service" (in this case, a version identified by the date 2006-08).

  There are times when the Discovery Service may need to know the logical type of service. In the case of Liberty defined services, we have adopted the convention of using a colon separated URN in the "liberty" namespace where the last element is the version identifier. In this case, the Discovery service could extract the logical type of service from the service type. However, that is just the Liberty convention and does not necessarily apply to service type definitions outside of the Liberty URN namespace.

  In non-Liberty-defined `<ServiceType>` URIs, the Discovery Service may understand the convention used in that particular namespace, or it may have some configuration defined knowledge of which URIs match to which logical type of service (perhaps via a user configurable parameter). If the DS cannot separate the logical type of service from the service version, the DS SHOULD treat the entire `<ServiceType>` URI as the logical type of service.

  The `<ServiceType>` URI is also typically used as the Namespace identifier for the XML schema for the service, so the version identifier typically shows up there as well – although this is NOT a normative requirement.

- **Framework Versioning** — the version of the communications framework used for ID-WSF messaging. Each ID-WSF message has a potential collection of SOAP headers defined by the various ID-WSF specifications which are tied together by the [LibertyIDWSF20SCR]. The [LibertySOAPBinding] specification defines the `<Framework>` element which carries a description of the framework. As of this release that consists primarily of a `version` attribute. [LibertyIDWSF20SCR] defines a particular version string to represent each concrete version of the specifications.

  The `Framework description` is included in ID-WSF messages, ID-WSF minted EPRs and in Discovery Service `<Query>` operations (in other words, the framework description is actively specified at each stage of the ID-WSF interaction model).

To ensure that the WSC communicates appropriately (from a versioning point of view) with the WSP, the WSC specifies both the service and framework versions that it supports during discovery and the Discovery Service matches the WSC capabilities with the appropriate registered service instances in order to return an EPR that the WSC can use.

2.3. ID-WSF Endpoint References (EPRs)

The general form of an EPR is illustrated in Example 1.

```
<wsa:EndpointReference ...>
  <wsa:Address>...some URI here...</wsa:Address>

  <wsa:ReferenceParameters>.....</ wsa:ReferenceParameters>

  <wsa:Metadata>...some metadata here... </wsa:Metadata>
</wsa:EndpointReference>
```

Example 1. General Form of an EPR

The EPRs are profiled, as specified below in Section 2.3.3, by placing Liberty-specific attributes and elements into the EPR. Specifically, a few attributes on the EPR itself and some sub-elements within `<wsa:Metadata>` element of the EPR. These Liberty-specific components are defined below in Section 2.3.1: EPR Profiling Attributes and Section 2.3.2: EPR Profiling Elements . These profiled EPRs are referred to as "ID-WSF EPRs", Example 2 illustrates an ID-WSF EPR.

Note
The use of these profiled EPRs does not necessarily replace WSDL; rather, they may be used either in conjunction with WSDL or without. This is also described in Section 2.3.3.

```
<wsa:EndpointReference
      notOnOrAfter="2005-08-15T23:18:56Z"
      ...>
  <wsa:Address>
    https://profile-provider.com/profiles/ someFoobarProfile
  </wsa:Address>

  <wsa:Metadata>
    <ds:Abstract>
      This is a personal profile containing common name information.
    </ds:Abstract>

    <ds:ProviderID>http://profile-provider.com/</ds:ProviderID>

    <ds:ServiceType>&PS1Namespace;</ds:ServiceType>

    <ds:Framework version="2.0" />

    <ds:SecurityContext>
      <ds:SecurityMechID>
        urn:liberty:security:2005-02:TLS:SAML
      </ds:SecurityMechID>

      <sec:Token>
        <!--  some security token goes here -->
      </sec:Token>
    </ds:SecurityContext>

    <ds:Options>
      <ds:Option>urn:liberty:id-sis-pp</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:cn</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:can</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:can:cn</ds:Option>
    </ds:Options>
  </wsa:Metadata>
</wsa:EndpointReference>
```

Example 2. An Instantiated ID-WSF EPR

2.3.1. EPR Profiling Attributes

This section defines the attributes that are used to profile EPRs as defined below in Section 2.3.3: ID-WSF Web Services Addressing EPR Profile . The full Discovery Service schema is given in Appendix A: Discovery Service Version 2.0 XSD .

2.3.1.1. wsu:Id — unique identifier for xml references to an EPR.

The `wsu:Id` attribute (Figure 1) is used when other elements in the XML document (e.g., message) need to refer to this EPR (for example, when this element is referenced in an XML signature).

```


     <!-- wsu:Id attribute for EPR - for xml document references to EPR -->

     <xs:attribute ref="wsu:Id" use="optional" />

```

Figure 1. `wsu:Id` — Schema Fragment

2.3.1.2. reqRef — request reference

The `reqRef` attribute (Figure 2) identifies which `<RequestedServiceType>` element in the Discovery Service `<Query>` request that this EPR was minted in response to. In other words this is used to associate the EPR in the `<QueryResponse>` with the `<RequestedServiceType>` in the `<Query>` request.

```

     <!-- Request Reference - to id which Request generated EPR -->

     <xs:attribute name="reqRef" type="xs:string" use="optional"/>

```

Figure 2. `reqRef` — Schema Fragment

2.3.1.3. `notOnOrAfter`

The `notOnOrAfter` attribute states the expiration timestamp for the EPR with which it is associated (Figure 3). See Example 2, above, for an instantiated EPR example.

Values of the `notOnOrAfter` attribute MUST be expressed in accordance with Liberty ID-WSF time value restrictions.

Liberty system entities SHOULD NOT rely on time resolution finer than milliseconds. Implementations MUST NOT generate time instants that specify leap seconds.

```

  <!-- EPR Expiration Timestamp  -->

  <xs:attribute name="notOnOrAfter" type="xs:dateTime"/>


```

Figure 3. `notOnOrAfter` — Schema Fragment

2.3.2. EPR Profiling Elements

This section defines the elements that are used to profile EPRs as defined below in Section 2.3.3: ID-WSF Web Services Addressing EPR Profile . The full Discovery Service schema is given in Appendix A: Discovery Service Version 2.0 XSD .

2.3.2.1. Abstract

The `<Abstract>` element (Figure 4) is used for conveying a textual, natural language description of the service instance.

```

  <!-- Abstract: natural-language description of service  -->

  <xs:element name="Abstract" type="xs:string"/>


```

Figure 4. `<Abstract>` — Schema Fragment

2.3.2.2. Provider ID

The `<ProviderID>` element (Figure 5) contains the URI of the provider of this service instance.

```

  <!-- Provider ID  -->

  <xs:element name="ProviderID" type="xs:anyURI"/>


```

Figure 5. `<ProviderID>` — Schema Fragment

2.3.2.3. Service Type

The `<ServiceType>` element (Figure 6) is used to identify a service type and version. This URI needs be constant across all implementations of a service to enable interoperability. Therefore, it is RECOMMENDED that this URI be the same as the targetNamespace URI of the abstract WSDL description for the service.

```

  <!-- Service Type -->

  <xs:element name="ServiceType" type="xs:anyURI"/>


```

Figure 6. `<ServiceType>` — Schema Fragment

Some examples of possible ServiceType URIs:

```
urn:liberty:disco:2006-08

urn:liberty:id-sis-pp:2003-08

http://myservices.com/gaming/1.0
```

2.3.2.4. `SecurityContext`

The `<SecurityContext>` element (Figure 7) is a container in which `<SecurityMechID>` elements and `<sec:Token>` elements are placed and thus associated with an ID-WSF EPR. The `<sec:Token>` element is used to either directly contain, or reference, security tokens and/or identity tokens.

Therefore, the `<SecurityContext>` element serves to denote the invocation context necessary for interacting with the service instance represented by the containing ID-WSF EPR.

NOTE: in some cases the DS will **not** be able to generate the necessary tokens to complete the security context. This will usually happen when a context needs a security token from a provider other than the DS (such as a non-related IdP). In such cases, the DS will include an empty token element with the `ref` attribute set to the following URI:

- `urn:liberty:disco:tokenref:ObtainFromIDP`

In such cases, the WSC receiving the EPR MUST communicate with the invoking principal's IdP's SSO Service (see [LibertyAuthn]) in order to obtain the necessary security token.

The value of the security mechanism in the security context will identify the type of security token that the WSC should request from the IdP. For example, if the security mechanism was "urn:liberty:...:SAMLV2," the WSC would know they needed a SAML 2.0 token with a subject confirmation of "...:holder-of-key" and would indicate so on the SSO Service request.

An ID-WSF EPR MAY contain more than one `<SecurityContext>` element. This serves to denote mutually-exclusive groupings of `<SecurityMechID>`s and `<sec:Token>`s, and thus different security contexts.

See Section 2.3.3: ID-WSF Web Services Addressing EPR Profile , below, for the precise specification of the mapping of `<SecurityContext>`, and its contents, to ID-WSF EPRs.

```

  <!--  Security Context Container  -->

  <xs:element name="SecurityContext">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="SecurityMechID"
                minOccurs="1"
                maxOccurs="unbounded"/>

        <xs:element ref="sec:Token"
                minOccurs="0"
                maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>


```

Figure 7. `<SecurityContext>` — Schema Fragment

See Example 2, above, for an instantiated ID-WSF EPR example, containing a `<SecurityContext>` element, itself containing `<SecurityMechID>` and `<sec:Token>` elements.

2.3.2.5. `SecurityMechID`

The `<SecurityMechID>` element (Figure 8) specifies the security mechanism(s) supported by the service instance represented by the ID-WSF EPR. These security mechanisms are represented as URIs, and are defined in [Liberty-SecMech].

The `<SecurityMechID>` element is used within the `<SecurityContext>` element described above. This is detailed in the Section 2.3.3: ID-WSF Web Services Addressing EPR Profile section below.

```
<!-- Security Mechanism ID -->

<xs:element name="SecurityMechID" type="xs:anyURI"/>
```

Figure 8. `<SecurityMechID>` — Schema Fragment

Some examples of possible SecurityMechID URI values (from [LibertySecMech]):

```
urn:liberty:security:2006-08:ClientTLS:SAMLV2

urn:liberty:security:2003-08:ClientTLS:SAML

urn:liberty:security:2006-08:TLS:SAMLV2
```

See Example 2, above, for an instantiated ID-WSF EPR example, containing a `<SecurityContext>` element containing `<SecurityMechID>` and `<sec:Token>` elements.

2.3.2.6. Framework

The `<Framework>` element (Figure 9) identifies the Liberty ID-WSF framework supported by the service instance at this endpoint. There MUST be at least one `<Framework>` element within an EPR.

Multiple `<Framework>` elements indicate that the service instance supports any of the specified ID-WSF versions at this same endpoint.

The structure and content of this element is defined in [LibertySOAPBinding].

```
<!-- Framework Description -->

<xs:element ref="sb:Framework" maxOccurs="unbounded" />
```

Figure 9. `<Framework>` — Schema Fragment

2.3.2.7. Action

The optional multi-occurence `<Action>` element (Figure 10) is used to identify the set of interfaces exposed by the provider at this endpoint.

Each `<Action>` element contains a URI that MUST match one of the `<wsa:Action>` URIs defined for the service.

When there are no `<Action>` elements in an EPR, the EPR can be used to invoke **all** of the interfaces for the defined service type.

This element is typically only included when the service instance specified in the EPR can only address a sub-set of the service's interfaces. A service instance may do this to scale their resources across different interfaces. For example, a service instance of the personal profile service may support the Query interface on a large cluster of systems, but require that the less frequently called, modify operations take place on some dedicated hardware.

```

  <!-- Action(s) - the interfaces available at this service -->

  <xs:element name="Action" type="xs:anyURI"  />

```

Figure 10. `<Action>` — Schema Fragment

2.3.2.8. Options

The `<Options>` element (Figure 11) expresses the "options" supported by a service instance. Thus they provide hints to a potential requester whether certain data or operations may be available with a particular service instance.

For example, an option may be provided stating that home contact information is available. If no Options element is present, it means only that the service instance does not advertise whether any options are available. Options may, in fact, be employed by the service instance. For example, it may be a simple service that is not capable of updating its entry in the Discovery Service when the available options change, so it avoids listing them at all. If the Options element is present, but is empty, it means that the service instance explicitly advertises that no options are available.

```

  <!-- Options -->

  <xs:element name="Options" type="OptionsType"/>

  <xs:element name="Option" type="xs:anyURI" />

  <xs:complexType name="OptionsType">
    <xs:sequence>
      <xs:element ref="Option" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>


```

Figure 11. `<Options>` — Schema Fragment

The `<Options>` element contains zero or more `<Option>` elements, each of which contains a URI identifying a particular option. The set of possible URIs for an `<Option>` element should be defined by the service type. For example, a person profile service specification would specify a set of options particular to its own domain. However, one common `<Option>` flag related to security, and thus common to ID-WSF services, is defined in Section 3.11: `Option` Value for Response Authentication .

2.3.3. ID-WSF Web Services Addressing EPR Profile

This section specifies the profile of WSA Endpoint References (EPRs). Profiling an EPR, yielding an ID-WSF EPR, is accomplished by placing various of the elements defined in Section 2.3.2: EPR Profiling Elements , above, into the EPR's `<wsa:Metadata>` element according to the rules defined below. All ID-WSF EPRs must adhere to the per-element rules in Section 2.3.2, and thereupon adhere to the rules defined in the following sections, depending upon the intended usage scenario for the ID-WSF EPR being minted.

For reference, the general form of an instantiated EPR is illustrated above in Example 1, and the `<wsa:EndpointReference>` schema fragment [WSAv1.0-SOAP] is illustrated below in Figure 12.

An ID-WSF EPR is normatively defined as a `<wsa:EndpointReference>` profiled as per this section.

Note

Except for the `<wsa:Address>` and `<wsa:ReferenceParameters>` elements, all elements discussed in the below sections are denoted as either being "absent" or "present" as content of the `<wsa:Metadata>` element of the ID-WSF EPR being minted.

```

  <xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>

  <xs:complexType name="EndpointReferenceType">
    <xs:sequence>
      <xs:element name="Address"
              type="tns:AttributedURIType"/>

      <xs:element name="ReferenceParameters"
              type="tns:ReferenceParametersType"
              minOccurs="0"/>

      <xs:element ref="tns:Metadata"
              minOccurs="0"/>

      <xs:any namespace="##other"
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>

    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
```

Figure 12. `<wsa:EndpointReference>` — Schema Fragment

2.3.3.1. ID-WSF EPR Minting Rules

ID-WSF EPRs are minted by both the Discovery Service (in response to `<Query>` requests) and by system entities acting as in a WSC or WSP role for inclusion in SOAP message header blocks such as the `<wsa:ReplyTo>` and the `<wsa:FaultTo>`, as discussed in [LibertySOAPBinding]. This section refers to these different parties collectively as "issuers."

The following rules MUST be observed by issuers when constructing an ID-WSF EPR:

1. A `notOnOrAfter` attribute MAY be present in each ID-WSF EPR. If absent, or if it has a value of *1970-01-01T00:00:00Z*, it means the issuer is not stipulating an expiration time for this ID-WSF EPR, and that its wielder is obliged to follow its own local policy for refreshing any cached copies. If present, the value should be set by the issuer according to local policy.

2. The value of the `<wsa:Address>` element MUST contain the endpoint address of the service instance being described by this EPR. This literally-addressed form of ID-WSF EPR is useful in order to ease the burden of WSCs from having to retrieve and parse WSDL in common cases. Additionally, the rules specified in Section 2.3.3.2: ID-WSF EPR Specifics MUST be adhered to.

3. A `wsu:Id` attribute MAY be present on the EPR root element.

4. A `reqRef` attribute MAY be present on the EPR root element.

5. Exactly one `<Abstract>` element MAY be present in the EPR `<Metadata>` element.

6. Exactly one <ProviderID> element MUST be present in the EPR <Metadata> element.
7. One or more <ServiceType> elements MUST be present in the EPR <Metadata> element.
8. One or more <Framework> elements MUST be present in the EPR <Metadata> element.
9. Optionally, one or more <Options> element(s). These are discussed in detail above, in Section 2.3.2.8.
10. Optionally, one or more <Action> element(s). These are discussed in detail above, in Section 2.3.2.7.
11. One or more <SecurityContext> elements SHOULD be present in each ID-WSF EPR. If so they, and their content, MUST adhere to the rules below, as well as the additional specific rules in Section 2.3.3.3: Security Mechanism Specifics :

    a. If no security or identity tokens are to be embedded, then place all the supported security mechanisms, denoted by <SecurityMechID> elements, in a single <SecurityContext> element.

    b. Else, if security and/or identity tokens are to be embedded or referenced (via <sec:Token> elements), then one MUST group corresponding <SecurityMechID> and <sec:Token> elements into the same <SecurityContext> element. In other words, all security and identity tokens within a <SecurityContext> element MUST apply to ALL of the security mechanisms in the same context.

    c. A security and/or identity token embedded in a <sec:Token> in a given ID-WSF EPR's <SecurityContext> element MAY be referenced from other <SecurityContext> elements, whether the other <SecurityContext> elements are contained within the given ID-WSF EPR or whether they are in another ID-WSF EPR in the list of ID-WSF EPRs being constructed.

       Such referencing is accomplished by using the ref attribute of a <sec:Token> element. When constructing such a reference, the referencing <sec:Token> MUST reference the <sec:Token> element containing the target embedded security token, as specified in [LibertySecMech].

    d. All <sec:Token> elements included in the <SecurityContext> element MUST have the usage attribute set to the appropriate value (as documented in [LibertySecMech]) indicating their intended purpose.

    e. If the issuer is unable to generate a necessary token, it MUST include an empty <sec:Token> element with the ref attribute set to the value urn:liberty:disco:tokenref:ObtainFromIDP

2.3.3.2. ID-WSF EPR Specifics

The information contained in an ID-WSF EPR is sufficient for making invocations for service instances. In other words, the information contained in this group together with the abstract WSDL specified by the ServiceType URI is sufficient to logically compute concrete WSDL with the rule set specified below.

The <wsa:Address> element of the ID-WSF EPR contains the URI of the endpoint. For SOAP-over-HTTP endpoints, the URI scheme MUST be "http" or "https."

Use of this addressing form implies <wsdl:binding> and <wsdl:service> elements according to the following rules (i.e., the concrete WSDL can be logically computed given the abstract WSDL and an ID-WSF EPR):

- The <wsdl:binding> contains a <wsdlsoap:binding> element. This specifies that the SOAP binding for WSDL is being used.
- The style attribute of the <wsdlsoap:binding> element is "*document*".
- The transport attribute of the <wsdlsoap:binding> element is *http://schemas.xmlsoap.org/soap/http*.

- The abstract WSDL corresponding to the `<ServiceType>` MUST contain a single `<portType>` element. The `<wsdl:binding>` element provides bindings for the operations specified in this `<wsdl:portType>`. Each operation binding includes an input element and an output element, each containing a single `<wsdlsoap:body>` element. The `use` attribute of the `<wsdlsoap:body>` elements is "literal".

- The `location` attribute of `<wsdlsoap:address>` is equal to `<wsa:Address>`.

- All other optional elements and attributes are not specified and thus default to the SOAP binding of WSDL.

2.3.3.3. Security Mechanism Specifics

With respect to `<SecurityMechID>` URIs: these URIs denote the security mechanisms supported by the service instance described by the ID-WSF EPR. Other specifications, such as [LibertySecMech] define the actual security mechanisms along with their identifying URIs. These security mechanisms refer to the way a WSC authenticates to a WSP ("peer-entity authentication") and/or provides message security ("data-origin authentication").

An ID-WSF EPR SHOULD list all of the security mechanisms that the service instance supports in order of preference. I.e. the most preferred security mechanism is first in the list, the next is the second-most preferred, and so on.

In the case that the set of supported security mechanisms varies with respect to endpoint address(es) and/or WSDL binding, the system entity constructing the ID-WSF EPRs MUST construct multiple ID-WSF EPRs with each ID-WSF EPR separately representing each supported mapping.

Also, any single `<SecurityMechID>` URI MUST NOT appear in more than one of the `<SecurityContext>` elements of any of the ID-WSF EPRs so constructed. In other words, each service instance may only specify one WSDL binding per supported security mechanism. If a sequence of ID-WSF EPRs is constructed, then the ID-WSF EPRs SHOULD appear in the order of the constructor's preference, and the `<SecurityContext>` elements within each should be in order of preference, as should the `<SecurityMechID>` elements within them—with the most preferred item listed first in each case.

For example: many web servers will require a different endpoint URI to be used for SOAP/HTTP clients authenticating using client TLS certificates than for clients which authenticate in some other fashion. See Example 4.

2.3.3.4. Action Specifics

With respect to `<Action>` URIs: these URIs denote the interfaces supported by the service instance described by the ID-WSF EPR. The service specific specifications, such as this document, define the actual interfaces along with their identifying URIs.

An ID-WSF EPR SHOULD NOT list actions unless the service instance at this endpoint does not support the complete set of service interfaces. In such a case, the ID-WSF EPR SHOULD list all of the available interfaces.

There is no preference or other significance to the ordering of the `<Action>` URIs.

2.3.3.5. Identity Invocation Context specifics

The invocation of an ID-WSF service can carry several identities as documented in [LibertySOAPBinding]. These identities include the `Sender`, the `InvocationIdentity`, the `TargetIdentity`, and the `Recipient`.

The Discovery Service, when minting ID-WSF EPRs, works to maintain the same identity invocation context that was used to invoke it such that the same logical `Sender`, `InvocationIdentity` and `TargetIdentity` are carried forth in messages invoked through the minted EPR. Of course, the `Recipient` of the subsequent invocation will be different as it will be the WSP to which this EPR points.

The Discovery Service generates security and/or identity tokens to convey these identities in the minted ID-WSF EPR. These tokens are placed into the `<sec:Token>` elements within `<SecurityContext>` element.

In preparing the necessary tokens to carry forth these identities, the Discovery Service may have to perform identity translations to obtain pseudonymous identifiers for the interested parties at the intended `Recipient`.

The rules for when and how the tokens are generated when the ID-WSF EPR is minted by the Discovery Service (in response to a DiscoveryQuery operation, see Section 3.3), are as follows:

- If the Principal, whose discovery resource is being queried, is the same as the invocation identity of the DiscoveryQuery operation — i.e., there is not a `<sb:TargetIdentity>` header block on the `<Query>` message — then the same effective invocation identity MUST be expressed by the Discovery Service's resultant selected security tokens for the invocation identity (which are embedded in `<sec:Token>` element(s) in the `<SecurityContext>` element in the ID-WSF EPR's `<wsa:Metadata>` element)
  Note
  Since the security tokens usually carry the identity of the `Sender` and that of the `InvocationIdentity` it is possible that a single `<SecurityContext>` may include multiple security tokens identifying each of the parties.

- Else, if the Principal, whose discovery resource is being queried, is not the same as the invocation identity of the DiscoveryQuery operation — i.e., a `<sb:TargetIdentity>` header block appears in the header of the `<Query>` message — then the invocation identity to be conveyed in the ID-WSF EPR is expressed as denoted in the bullet item above, and additionally, a identity token denoting the target identity (per [LibertySecMech] and [LibertySecMech20SAML]) is also embedded in a `<sec:Token>` element in the `<SecurityContext>` element in the ID-WSF EPR's `<wsa:Metadata>` element.

The rules for when and how the above identity tokens are included as above when the ID-WSF EPR is minted by a WSC or WSP (refer to Section 2.3.3.1, above, for context), are as follows:

- If the intended target identity is to be the same as that of the intended invocation identity, then the intended invocation identity MUST be expressed in the minted ID-WSF EPR as detailed in the rules above (first bullet item).

- If the intended target identity is to be different than the intended invocation identity, then the intended invocation identity and the intended target identity both MUST be expressed in the minted ID-WSF EPR as detailed in the rules above (second bullet item).

The recipient of an ID-WSF EPR distinguishes between the various tokens contained within a `<sec:Token>` element via the `usage` attribute as follows:

- A token with the `usage` attribute set to `urn:liberty:security:tokenusage:2006-02:SecurityToken` contains a security token that MUST be placed into the `<wsse:Security>` header block (according to [Liberty-SecMech] and its related profiles) when a message is generated for the target of the ID-WSF EPR.

  If multiple `<sec:Token>`s are included in a single `<ds:SecurityContext>`, they MUST ALL be placed into the same `<wsse:Security>` header block.

- A token with the `usage` attribute set to `urn:liberty:security:tokenusage:2006-08:TargetIdentity` contains an identity token that MUST be placed into the `<sb:TargetIdentity>` header block (according to [LibertySOAPBinding]) when a message is generated for the target of the ID-WSF EPR.

2.3.4. Effective Web Services Addressing EPR

The net effect of the ID-WSF profile of the EPR is as if the `EndpointReferenceType` were defined with the schema fragment below. There are several things to note about this schema including:

- There is no normative XML schema defined as such, this is just an approximation of what the schema could look like.

- While the elements within the `<Metadata>` element appear to be ordered, they can all appear in any order and can have other elements appear between the listed elements. This is why they are contained within a multi-occurence `<xs:choice>`.
- Four attributes have been added to the EPR element itself: the notOnOrAfter timestamp and three different IDs (for use in different circumstances).
- Seven sub-elements were added to the `<Metadata>` element.
- The `<Metadata>` sub-element `<disco:ProviderID>` MUST appear exactly once in an ID-WSF EPR, even though the schema below does not enforce that requirement (because of a limitation in XML Schema – or perhaps in the author's understanding of XML schema).

  The `<Metadata>` sub-elements: `<sbf:Framework>`, and `<disco:ServiceType>` MUST appear at least once in an ID-WSF EPR, even though the schema below does not enforce that requirement (because of a limitation in XML Schema – or perhaps in the author's understanding of XML schema).

```

...
<xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>
<xs:complexType name="EndpointReferenceType" mixed="false">
  <xs:sequence>
    <xs:element name="Address" type="tns:AttributedURIType"/>
    <xs:element name="ReferenceParameters"
            type="tns:ReferenceParametersType" minOccurs="0"/>
    <xs:element ref="tns:Metadata" minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax"
             minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="notOnOrAfter" type="xs:dateTime" use="optional" />
  <xs:attribute ref="wsu:Id" use="optional" />
  <xs:attribute name="reqRef" type="xs:string" use="optional" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="ReferenceParametersType" mixed="false">
  <xs:sequence>
    <xs:any namespace="##any" processContents="lax"
              minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="Metadata" type="tns:MetadataType"/>
<xs:complexType name="MetadataType" mixed="false">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="disco:Abstract" minOccurs="0" />
      <xs:element ref="sbf:Framework" maxOccurs="unbounded"/>
      <xs:element ref="disco:ProviderID" />
      <xs:element ref="disco:ServiceType" maxOccurs="unbounded"/>
      <xs:element ref="disco:SecurityContext" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="disco:Options" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="disco:Action" minOccurs="0" maxOccurs="unbounded" />
      <xs:any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

```

Example 3. Effective ID-WSF EPR Schema

2.3.5. Example Liberty ID-WSF EPRs

```
<wsa:EndpointReference
      notOnOrAfter="2005-08-15T23:18:56Z"
       ...>
  <wsa:Address>
    http://profile-provider.com/profiles/someFoobarProfileAddr
  </wsa:Address>

  <wsa:Metadata>
    <ds:Abstract>
      This is a personal profile containing common name information.
    </ds:Abstract>

    <ds:ProviderID>http://profile-provider.com/</ds:ProviderID>

    <ds:ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>

    <sbf:Framework version="2.0" />

    <ds:SecurityContext>
      <ds:SecurityMechID>
        urn:liberty:security:2006-08:ClientTLS:SAMLV2
      </ds:SecurityMechID>

      <ds:SecurityMechID>
        urn:liberty:security:2005-02:ClientTLS:SAML
      </ds:SecurityMechID>

      <sec:Token wsu:id="_10"
          usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
        <!--  some security token goes here -->
      </sec:Token>
    </ds:SecurityContext>

    <ds:SecurityContext >
      <ds:SecurityMechID>
        urn:liberty:security:2005-02:ClientTLS:X509
      </ds:SecurityMechID>

      <sec:Token wsu:id="_20"
        usage="urn:liberty:security:tokenusage:2006-08:InvocationIdentity">
        <!-- Identity Token goes here -->
      </sec:Token>
    </ds:SecurityContext>

    <ds:Options>
      <ds:Option>urn:liberty:id-sis-pp</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:cn</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:can</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:can:cn</ds:Option>
    </ds:Options>
  </wsa:Metadata>
</wsa:EndpointReference>

<wsa:EndpointReference
      notOnOrAfter="2005-08-15T23:18:56Z"
      ...>
  <wsa:Address>
    http://profile-provider.com/profiles/anotherFoobarProfileEndpointAddr
  </wsa:Address>

  <wsa:Metadata>
    <ds:Abstract>
      This is a personal profile containing common name information.
    </ds:Abstract>

    <ds:ProviderID>http://profile-provider.com/</ds:ProviderID>

    <ds:ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>

    <sb:Framework version="2.0" />

    <ds:SecurityContext>
      <ds:SecurityMechID>
        urn:liberty:security:2006-08:TLS:SAMLV2
      </ds:SecurityMechID>

      <sec:Token ref="_10" usage="urn:liberty:security:tokenusage:2006-08:SecurityToken" />
    </ds:SecurityContext>

    <ds:Options>
      <ds:Option>urn:liberty:id-sis-pp</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:cn</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:can</ds:Option>
      <ds:Option>urn:liberty:id-sis-pp:can:cn</ds: Option>
    </ds:Options>
  </wsa:Metadata>
</wsa:EndpointReference>
```

Example 4. Instantiated List of ID-WSF EPRs Illustrating Multiple `<SecurityContext>` Elements with both Embedded and Referenced `<sec:Token>` Elements

2.4. Service Metadata

The discovery Service mints the ID-WSF EPRs described in the previous section using information provided by the WSP in the WSP's registered Service Metadata.

2.4.1. Service Metadata element

The Service Metadata is used to describe a single instance of a service hosted by a WSP as it applies to all principals (i.e., the principal independent information related to an instance).

This single instance can include multiple endpoints, multiple security mechanisms, and even multiple service types. Multiple service types SHOULD only be included in a single Service Metadata element if the WSP considers those service types to be different versions of the same service (for example, *urn:liberty:disco:2006-08* and *urn:liberty:disco:2003-08* are two different versions of the Liberty ID-WSF Discovery Service).

Most of the fields present in the Service Metadata have the same purpose and meaning as the elements of the same name in the ID-WSF EPR (as this is where the Discovery service gets those elements for the ID-WSF EPR).
When fields permit multiple values, the order of entries in the SvcMD **is** significant with higher preference items coming first. This comes into plan should the WSC request a subset of the possible results when querying the Discovery Service (in which case the entries with the higher preference – those listed first – would be used to mint the ID-WSF EPRs in the response).

```

  <!-- Service Metadata (SvcMD) - metadata about service instance -->

  <xs:element name="SvcMD" type="SvcMetadataType"/>
  <xs:complexType name="SvcMetadataType">
    <xs:sequence>
      <xs:element ref="Abstract"                     />
      <xs:element ref="ProviderID"                   />
      <xs:element ref="ServiceContext"  maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="svcMDID" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- ServiceContext - describes service type/option/endpoint context -->
  <xs:element name="ServiceContext" type="ServiceContextType"/>
  <xs:complexType name="ServiceContextType">
    <xs:sequence>
      <xs:element ref="ServiceType"    maxOccurs="unbounded" />
      <xs:element ref="Options"      minOccurs="0"
                              maxOccurs="unbounded" />
      <xs:element ref="EndpointContext" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- EndpointContext - describes endpoints used to access service -->
  <xs:element name="EndpointContext" type="EndpointContextType" />
  <xs:complexType name="EndpointContextType">
    <xs:sequence>
      <xs:element ref="Address"      maxOccurs="unbounded" />
      <xs:element ref="sbf:Framework"  maxOccurs="unbounded" />
      <xs:element ref="SecurityMechID" maxOccurs="unbounded" />
      <xs:element ref="Action"      minOccurs="0"
                             maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- SvcMD ID element used to refer to Service Metadata elements -->
  <xs:element name="SvcMDID" type="xs:string" />

```

Figure 13. Service Metadata — Schema Fragment

2.4.1.1. svcMDID

The `svcMDID` attribute is a unique identifier assigned by the Discovery Service during service metadata registration and used on later principal registrations.

The value of the identifier MUST be unique across all registered service metadata for the registering WSP at the DS and MAY be unique across all WSPs.

2.4.1.2. Abstract

A text description of the service.

2.4.1.3. ProviderID

The URI of the provider of this service instance.

2.4.1.4. ServiceContext

The `<ServiceContext>` describes the set of service versions and options that are available at a particular set of endpoints. A Service Metadata description may have multiple `<ServiceContext>`s when they support a particular version (or set of options) of the service at one set of endpoints and another version at a different set of endpoints.

The elements contained within a `<ServiceContext>`s are discussed below:

2.4.1.4.1. ServiceType

The URI of which defines the type of service.

Note that there may be multiple service types defined in a service metadata indicating that multiple distinct services are available at the same endpoint. This typically occurs when multiple versions of the same general type of service are available at the same endpoint although it is possible that very different services could be at the same endpoint.

2.4.1.4.2. Option

The Option(s) supported by this service instance.

Multiple options may be specified indicating that this service instance supports all of the listed options.

2.4.1.4.3. EndpointContext

While not explicitly in an ID-WSF EPR, the contents of this element show up in various locations within the IPR and/or guide the generation of the contents of the EPR.

Multiple `<EndpointContext>` elements may appear if the same service is available via different, incompatible combinations of the contents (such as a TLS and a non-TLS endpoint at different addresses).

The sub-elements include:

2.4.1.4.3.1. Address

A URI describing the address to which messages should be sent to communicate with this provider.

If multiple addresses are specified they are all considered equally valid addresses for this same service (such that if a Discovery Service were to mint all of the possible EPRs for this case, there would be a separate EPR for each address specified since an EPR can only include a single address).

In the case where the Discovery service has been asked to mint a subset of the possible EPRs (see Section 3.3), the Discovery service is free to select any of the specified addresses using whatever local policy it chooses.

2.4.1.4.3.2. Framework

The SOAP Bindings ([LibertySOAPBinding]) `<sbf:Framework>` element describing the version of the ID-WSF framework supported at this endpoint..

Multiple `<Framework>` elements may be specified if they can be used at each of the `<Address>` URIs within this `<EndpointContext>`.

2.4.1.4.3.3. SecurityMechID

The Security Mechanism URI(s) (defined in [LibertySecMech] and its related profiles) supported by this endpoint.

Multiple `<SecurityMechID>` elements may be specified indicating that any of these mechanisms can be used at this endpoint.

Note that while a particular security mechanism may need a particular form of a security token, the registering WSP cannot provide such tokens. It is up to the Discovery service to mint the necessary token, or indicate to the WSC that they need to obtain the token from their IdP.

2.4.1.4.3.4. Action

The URI indicating the supported service action at this endpoint. This is typically used when only a sub-set of the entire service's operations are available at this endpoint.

Multiple `<Action>` elements may be specified to indicate that there are multiple operations available at this endpoint.

If no `<Action>` element is specified, all service operations are available at this endpoint.

2.4.2. Minting ID-WSF EPRs based upon Service Metadata

Service Metadata is stored in the Discovery Service in order to guide the minting of ID-WSF EPRs by the Discovery Service in response to queries from WSCs.

One can visualize that the entire set of elements within a single Service Metadata can result in a large number of possible EPRs based upon the possible combinations of those elements.

The Discovery Service MUST mint ID-WSF EPRs as if the following process took place (there is NOT a normative requirement to implement this exact process, just a requirement that the results generated by whatever process is used by the DS MUST result in the set of data that would result from this process).

1. Eliminate portions of the Service Metadata that do not conform to the search requirements (such as unsupported (by the WSC) security mechanisms or framework versions, or undesired service types).

2. If an `<EndpointContext>` element had all occurences of a given sub-element (such as `<Framework>`) eliminated, eliminate the context.

3. For each remaining `<Address>`) element within a remaining `<EndpointContext>` element, an EPR SHOULD be minted.

4. For each EPR, assign one `<Address>`) element to the `<wsa:Address>`) element in the EPR and use the rest of the `<EndpointContext>` that contained this address to build the necessary `<Metadata>` sub-elements for the ID-WSF EPR (. `<SecurityContext>`(s), `<Action>`(s), `<Framework>`(s), etc.).

5. Fill out the rest of the ID-WSF EPR using the service wide elements (`<Abstract>`, `<ProviderID>`, `<ServiceType>`(s), etc.).

6. If necessary, generate any security and/or identity tokens and place them into the appropriate `<SecurityContext>` element(s).

The set of EPRs generated by this process may be further restricted by the request parameters on the DiscoveryQuery operation, see Section 3.3).

2.4.3. Service Metadata Example

Some examples to help show how service metadata works.

2.4.3.1. A simple service

This is an example of a simple service that has a single endpoint, supports a single framework version (2.0), and only supports a single security mechanism.

```
<ds:svcMD svcMDID="1234">
   <ds:Abstract>This is a simple service metadata definition</ds:abstract>
   <ds:ProviderID>http://simpler.providers.com</ds:ProviderID>
   <ds:ServiceContext>
      <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
      <ds:EndpointContext>
         <ds:Address>https://simple.providers.com/PP</ds:Address>
         <sb:Framework version="2.0"/>
         <ds:SecurityMechID>
            urn:liberty:security:2003-08:TLS:Bearer
         </ds:SecurityMechID>
      </ds:EndpointContext>
   </ds:ServiceContext>
</ds:SvcMD>
```

Figure 14. Service Metadata example: A simple service

2.4.3.2. A complex service

This is an example of a service metadata definition with a number of complex attributes including:

- Multiple service versions **and** multiple framework versions on the same endpoint.
- There are two service contexts, one for one version of the service and one for a different version of the service. So, for example, the 2003-08 version of the service is only available at the URL *https://old.providers.com/PP* and only for framework version *1.1*
- Multiple interfaces on different endpoints with different security mechanisms
- There are multiple, redundant, addresses for the TLS endpoint for the *2007-11* version of the service.

```
<!-- Service Metadata Example: A complex service -->
<ds:SvcMD svcMDID="4567">
   <ds:Abstract>This is a complex service metadata definition</ds:abstract>
   <ds:ProviderID>http://complex.providers.com</ds:Provide rID>
   <ds:ServiceContext>>
      <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
      <ds:EndpointContext>
         <ds:Address>https://old.providers.com/PP</ds:Addr ess>
         <sb:Framework version="1.1" />
         <ds:SecurityMechID>
            urn:liberty:security:2003-08:TLS:Be arer
         </ds:SecurityMechID>
      </ds:EndpointContext>
   </ds:ServiceContext>>
   <ds:ServiceContext>>
      <ds:ServiceType>urn:liberty:pp:2007-11</ds:ServiceType >
      <ds:EndpointContext>
         <ds:Address>https://complex.provider s.com/PP</ds:Address>
         <ds:Address>https://backup.complex.providers.com/ PP</ds:Address>
         <sb:Framework version="2.0" />
         <ds:SecurityMechID>
            urn:liberty:security:2003-08:TLS:Bearer
         </ds:SecurityMechID>
      </ds:EndpointContext>
      <ds:EndpointContext>
         <ds:Address>http://complex.providers.c om/PP</ds:Address>
         <sb:Framework version="2.0" />
         <ds:SecurityMechID>
            urn:liberty:security:2003-08:null:SAMLV2
         </ds:SecurityMechID>
      </ds:EndpointContext>
   </ds:ServiceContext>>
</ds:SvcMD>
```

Figure 15. Service Metadata example: A complex service

2.4.3.3. Another complex service

This is an example of a service metadata definition where the service has some of its operations at one endpoint and others at a different endpoint (thus splitting the service operations across different instances).

This service is still defined with a single service context since the endpoints all expose the same service type.

```
<!-- Service Metadata Example: A simple service -->
<ds:SvcMD svcMDID="8901">
   <ds:Abstract>Another example complex service</ds:abstract>
   <ds:ProviderID>http://split.providers.com</ds:ProviderID>
   <ds:ServiceContext>>
      <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
      <ds:EndpointContext>
         <ds:Address>https://cluster1.split.providers.com/PP</ds:Address>
         <sb:Framework version="2.0" />
         <ds:SecurityMechID>
            urn:liberty:security:2003-08:TLS:Bearer
         </ds:SecurityMechID>
         <ds:Action>urn:liberty:pp:2003-08:Query</ds:Action>
      </ds:EndpointContext>
      <ds:EndpointContext>
         <ds:Address>https://writer.split.providers.com/PP</ds:Address>
         <sb:Framework version="2.0" />
         <ds:SecurityMechID>
            urn:liberty:security:2003-08:TLS:Bearer
         </ds:SecurityMechID>
         <ds:Action>urn:liberty:pp:2003-08:Modify</ds:Action>
      </ds:EndpointContext>
   </ds:ServiceContext>>
</ds:SvcMD>
```

Figure 16. Service Metadata example: Another complex service

3. Discovery Service

A Discovery Service is a web service providing both identity based and non-identity based operations.

The identity based Discovery Service interfaces facilitate requesters' discovery of identity service instances on a per-identity basis, and acquisition of ID-WSF Endpoint References (ID-WSF EPRs) "pointing" to the discovered service instances. These ID-WSF EPRs provide requesters with the information necessary to invoke discovered service instances.

The non-identity based Discovery Service interfaces provide a WSP with principal-independent management of their metadata stored at the Discovery Service (which is, through an identity-based interface, associated with a principal).

Thus in an abstract sense, the Discovery Service is essentially a web service interface to per-identity "discovery resources", each of which can be viewed as a registry of ID-WSF EPRs. The notion of "discovery resources" is an abstract way of referring to what are concretely "identity-indexed Discovery Service instances".

The Discovery Service can also be used as a non-identity service to discover and obtain ID-WSF EPRs for non-identity services. For example, the Discovery Service could be used to locate the available Authentication Services before a principal identity has been established.

Entities can register ID-WSF EPRs, pointing to their identity services, with a discovery resource, and this will allow other entities to discover them. A common use case is that a Principal places references (aka ID-WSF EPRs) to his or her personal profile, calendar, and so on, in a discovery resource so that they may be discovered by other entities, e.g., web service providers who wish to provide the Principal with value-added services.

When invoked as an identity service, the act of discovering service instances is implicitly on a per-identity basis. This occurs in a number of fashions in ID-WSF including:

- When a Principal authenticates to a service provider using a SAMLv2 profile (or similarly via ID-FF), the identity provider conveys, within the authentication assertion, an ID-WSF EPR pointing explicitly to the *Principal's* discovery service resource, which the SP may then use to discover the Principal's various services.

- A Principal's (LUAD-)WSC authenticates via the Authentication Service (see [LibertyAuthn]), which will likely return an ID-WSF EPR for the Principal's Discovery Service resource.

- Any Identity Token (see [LibertySecMech]), or security token may contain a Discovery Service bootstrap ID-WSF EPR (see Section 4: Discovery Service ID-WSF EPR conveyed via a Security Token) which contains the necessary information to access the Principal's Discovery Service resource.

The Discovery service is identified by ID-WSF EPRs, which themselves have been crafted (typically by an identity provider) such that they identify the discovery service resource (aka Discovery Service instance) mapped to the Principal in question.

The Discovery Service is intended to be used in conjunction with other ID-WSF specifications. For example, security mechanisms are not specified here, because they are defined in [LibertySecMech]. At the same time, the Discovery Service is specified such that it could be used with other security mechanisms, not yet defined.

The Discovery Service is designed to be describable by WSDL [WSDLv1.1], and an abstract WSDL definition is included in this document, see Appendix B: Discovery Service WSDL . This WSDL document defines two "WSDL operations" for the Discovery Service. The first is the *DiscoveryQuery* operation. This operation returns an enumeration of ID-WSF EPRs for a given search criteria.

To enforce access control policies, security tokens may need to be presented by the client when interacting with a Discovery Service instance. While the definition of these security tokens is outside the scope of this specification, it is common for the same provider that is hosting the Discovery Service to also be the entity that generates the security tokens necessary to access the service. To avoid extra network round-trips, arrangements are made here so that security tokens may be provided as part of the Discovery Service lookup response.

3.1. Service URIs

Table 1. Discovery Service URIs

Use	URI
Service Type	*urn:liberty:disco:2006-08*
Query wsa:Action	*urn:liberty:disco:2006-08:Query*
QueryResponse wsa:Action	*urn:liberty:disco:2006-08:QueryResponse*
SvcMDAssociationAdd wsa:Action	*urn:liberty:disco:2006-08:SvcMDAssociationAdd*
SvcMDAssociationAddResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDAssociationAddResponse*
SvcMDAssociationQuery wsa:Action	*urn:liberty:disco:2006-08:SvcMDAssociationQuery*
SvcMDAssociationQueryResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDAssociationQueryResponse*
SvcMDAssociationDelete wsa:Action	*urn:liberty:disco:2006-08:SvcMDAssociationDelete*
SvcMDAssociationDeleteResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDAssociationDeleteResponse*
SvcMDQuery wsa:Action	*urn:liberty:disco:2006-08:SvcMDQuery*
SvcMDQueryResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDQueryResponse*
SvcMDRegister wsa:Action	*urn:liberty:disco:2006-08:SvcMDRegister*
SvcMDRegisterResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDRegisterResponse*
SvcMDReplace wsa:Action	*urn:liberty:disco:2006-08:SvcMDReplace*
SvcMDReplaceResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDReplaceResponse*
SvcMDDelete wsa:Action	*urn:liberty:disco:2006-08:SvcMDDelete*
SvcMDDeleteResponse wsa:Action	*urn:liberty:disco:2006-08:SvcMDDeleteResponse*

3.2. Status Codes

The following status code strings are defined:

- *OK*: message processing succeeded
- *Failed*: general failure code
- *Forbidden*: the request was denied based on policy
- *Duplicate*: the request was denied because it would result in duplicate data in the service
- *LogicalDuplicate*: the request was denied because it would result in logically duplicate data in the service
- *NoResults*: the query had no matching results
- *NotFound*: the specified item(s) were not found

These strings are expected to appear in the "code" attribute of `<Status>` elements used in SOAP-bound Discovery Service protocol messages [LibertySOAPBinding]. Specific uses for the status codes are defined in the processing rules for individual messages. The "ref" attribute on the `<Status>` element is not used in this specification, so it MUST NOT appear on Status elements in Discovery Service protocol messages. The contents of the `comment` attribute are not defined by this specification, but it may be used for additional descriptive text intended for human consumption (for example, to carry information that will aid debugging).

3.3. Operation: *DiscoveryQuery*

The *DiscoveryQuery* WSDL operation enables a requester to obtain an enumeration of ID-WSF EPRs (see Section 2: Discovery Service Information Model) — the requester sends a `<Query>` message and receives a `<QueryResponse>` message in return. Also, because a provider hosting a Discovery Service may also be playing other roles on behalf of Principals (such as a *Policy Decision Point* or an *Authentication Authority*), the *DiscoveryQuery* operation can also function as a security token service, providing the requester with an efficient means of obtaining security tokens that may be necessary to invoke service instances described in the `<QueryResponse>`.

3.3.1. `wsa:Action` values for `DiscoveryQuery` Messages

`<Query>` messages MUST include a `<wsa:Action>` SOAP header with the value of `urn:liberty:disco:2006-08:Query`.

`<QueryResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of `urn:liberty:disco:2006-08:QueryResponse`.

3.3.2. `<Query>` Message

A `<Query>` request is an attempt to retrieve ID-WSF EPRs suitable for use in the same identity context that was used to make the request. In particular, the Target Identity (See [LibertySOAPBinding]), if applicable, is used to restrict the results to just those for the specified principal. The Invocation Identity will be verified against an access control list to ensure that they have access to the requested results.

A `<Query>` request message is empty in the minimal case. Such a request indicates the requester is requesting all available ID-WSF EPRs, regardless of security mechanisms or service types. The result set is dependant upon the local access control policies of the discovery service instance.

Alternatively, a request can be qualified with a set of `<RequestedService>` elements, which enables the requester to specify that all ID-WSF EPRs returned must be offered via one or more service instances complying with the specified search criteria. For each `<RequestedService>` specified, the requester specifies the search criteria for the DS to use in determining if there is a matching instance. The search criteria includes zero or more of any of the following:

- `<ServiceType>` the requested type of service. Service Type URIs are defined by the individual service specifications and contain both service class and service versioning information.

  Multiple `<ServiceType>`s MAY be specified in a single `<RequestedService>`s element in order to allow the WSC to specify what the WSC considers to be different versions of the same service.

  When multiple entries are listed, the order of such entries is an indication of the preference as to which `<ServiceType>` the WSC would prefer to see in the results, with the first being the most preferred. This typically only impacts a request where the WSC indicates that they only want a subset of the results returned (see `resultsType` below).

  When a request results in multiple ID-WSF EPRs in a response, the preference order specified by the WSC on the request MAY have no impact on the order of results returned by the Discovery Services. The Discovery Service is free to return the results of the request in whatever order it chooses.

  If not specified, then any service instance would be considered a match for this criteria.

  A service instance must support at least one of the specified service types in order to be considered a match for this criteria.

- `<ProviderID>` the requested provider ID(s). This is used when the WSC wants to communicate with a particular WSP. Frequently such requests are made without specifying a `<ServiceType>` element in the request, but doing so is not prohibited.

  If not specified, then any service instance would be considered a match for this criteria.

  A service instance must contain at least one of the specified providerIDs in order to be considered a match for this criteria.

  The order of the `<ProviderID>` elements is an indication as to the preference of the requester with the first such element being the most desired (declining preference order). The Discovery Services is free to return the results of the request in whatever order it chooses.

- `<Options>` — an optional multi-occurence element defining options SETs desired for the service.

  If not specified, any service instance will be considered a match for this criteria.

  An option SET is defined within each `<Options>` element and contains a list of the desired options. The service instance MUST support ALL of the options within the option SET in order to be considered a match for this request.

  If more than one `<Options>` element is specified (thus defining multiple option SETs), service instances that match ANY of the SETS are considered a match for this request. As noted above, to match a SET, you have to match ALL of the entries within the SET.

  Service instance EPRs registered without an `<Options>` element are always considered a match from the point of view of any possible `<Options>` search criteria.

  The order of the `<options>` elements is an indication as to the preference of the requester with the first such element being the most desired (declining preference order). The Discovery Services is free to return the results of the request in whatever order it chooses.

- `<SecurityMechID>` - an optional multi-occurence element specifying the security mechanism identifier(s) (see [LibertySecMech]) that the WSC is willing to use to invoke the WSP. If not specified, any security mechanism registered for a service will be considered a match for this criteria.

  A service instance MUST support at least one of the requested security mechanisms in order to be considered a match for this request.

  The order of the `<SecurityMechID>` elements is an indication as to the preference of the requester with the first such element being the most desired (declining preference order). The Discovery Services is free to return the results of the request in whatever order it chooses.

- `<Framework>` — an optional multi-occurence element specifying the framework description(s) supported by the WSC.

  If not specified, the Discovery Service SHOULD use the value of the framework description used in the ID-WSF framework layer for the current request (e.g., if the call to the Discovery Service was made using an ID-WSF version 2.0 message the request SHOULD be treated as if a `<disco:Framework>` element was present and contained the value specified in the `<sbf:Framework>` SOAP header.

  Multiple `<disco:Framework>` elements MAY be specified, indicating that the WSC has the capability to support ANY of the specified versions. The order of elements in such a case indicates the WSC's preference with the most preferred coming first.

  Note that while both the `<disco:Framework>` and the `<sbf:Framework>` elements are of the same type (`<sbf:FrameworkType>`), the elements themselves are in different namespaces. The element within the `<RequestedService>` is in the Discover Service Namespace, while the element within any ID-WSF EPRs and the SOAP header block on an ID-WSF message are in the SOAP Bindings namespace.

- `<Action>` — an optional multi-occurence element specifying the `wsa:Action` value(s) for the interfaces of the service that the WSC intends to make use of.

  If not specified, the Discovery Service SHOULD treat this request as a request for all of the interfaces at the requested specified instance.

  Unlike the other sub-elements of the `<RequestedService>` element, if multiple `<Action>` elements are specified it indicates that the WSC intends to invoke **all** of the specified interfaces and the Discovery Service SHOULD return the set of EPRs that are necessary to reach the complete set of specified interfaces.

  Services registered without an `<Action>` element (which is the norm) are treated as exposing **all** of the interfaces defined for that type of service.

The Discovery Service will return the set of EPRs for service instances that intersect with the search criteria specified in the `<RequestedService>` element. This may result in a single EPR in the response or it may result in a multitude of EPRs, depending upon the search criteria and the service instance definitions available (see Section 2.3.3: ID-WSF Web Services Addressing EPR Profile).

The result set of EPRs generated in response to a particular `<RequestedService>` element can be further controlled using the following attributes:

- `reqID` — an optional attribute identifying this `<RequestedService>` request. Typically only used when multiple `<RequestedService>` elements are included in a single Discovery Service `<Query>`.

  If present the value of this attribute will be placed into the `reqRef` attribute in any EPRs that result from this `<RequestedService>` element (see Section 2.3.1.2 above).

  The value of `reqID` SHOULD be different for all `<RequestedService>` elements in a given `<Query>`.

- `resultsType` — an optional attribute describing the results desired by the requestor. This value may be set to:

  - **best** — the Discovery Service SHOULD return the smallest set of EPRs while still meeting the minimum requirements of the request. This will typically be a single EPR.

  - **all** — the Discovery Service SHOULD return all of the matching entries for the given search criteria.

    This would typically be used when the client wants to choose which EPRs within the DS database it should use.

    This option should be used with caution as it can cause the DS to perform substantial work in order to mint all of the matching EPRs and the necessary security tokens for those EPRs.

  - **only-one** — a restricted version of **best** which further restricts the resulting output to **exactly one** EPR, even if the minimum requirements of the request would require multiple EPRs. A client would typically specify this option if it was going to ignore anything other than the first EPR returned.

  If `resultsType` is not specified the Discovery Service may make its own determination (under local policy) as to which set of results to return.

Requestors SHOULD include at least one `<ServiceType>` or `<ProviderID>` element, and MAY include any number of both of them.

Requesters SHOULD construct a Query to be as qualified as possible, as the Discovery Service instance may have to perform significant work for each item in the result set, especially if security tokens will be generated.

```

  <!-- Query Message Element & Type -->

  <xs:element name="Query" type="QueryType"/>

  <xs:complexType name="QueryType">
    <xs:sequence>
      <xs:element name="RequestedService"
              type="RequestedServiceType"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>

    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="RequestedServiceType">
    <xs:sequence>
      <xs:element ref="ServiceType" minOccurs="0" maxOccurs="unbounded" />

      <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded" />

      <xs:element ref="Options" minOccurs="0" maxOccurs="unbounded"/>

      <xs:element ref="SecurityMechID" minOccurs="0" maxOccurs="unbounded"/>

      <xs:element ref="Framework" minOccurs="0" maxOccurs="unbounded"/>

      <xs:element ref="Action" minOccurs="0" maxOccurs="unbounded"/>

      <xs:any namespace="##other"
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>

    </xs:sequence>

    <xs:attribute name="reqID" type="xs:string" use="optional" />
    <xs:attribute name="resultsType" type="xs:string" use="optional" />

  </xs:complexType>
```

Figure 17. Query Message — Schema Fragment

```

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    <Query xmlns="&DS2Namespace;">
      <RequestedService>
        <ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>

        <SecurityMechID>urn:liberty:security:2006-08:ClientTLS:SAMLV2</SecurityMechID>
        <SecurityMechID>urn:liberty:security:2005-02:ClientTLS:SAML</SecurityMechID>
        <SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</SecurityMechID>
        <Framework version="2.0" />

      </RequestedService>
    </Query>
  </soap:Body>
</soap:Envelope>

```

Example 5. SOAP message containing a Query

3.3.3. `QueryResponse`

A `<QueryResponse>` message conveys the results of the query as a set of ID-WSF EPRs, i.e., profiled `<wsa:EndpointReference>` elements (see Section 2.3.3: ID-WSF Web Services Addressing EPR Profile).

As specified in Section 2.3.3, security tokens, appropriate for subsequent invocation(s) of the service instances represented by the returned ID-WSF EPRs, MAY be provided within the ID-WSF EPRs in the response.

A status code is also included in the response.

```

  <!-- QueryResponse Message Element & Type -->

  <xs:element name="QueryResponse" type="QueryResponseType"/>

  <xs:complexType name="QueryResponseType">
    <xs:sequence>
      <xs:element ref="lu:Status"/>

      <xs:element ref="wsa:EndpointReference"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>


```

Figure 18. `<QueryResponse>` — Schema Fragment

An example SOAP message containing a `<QueryResponse>` message is illustrated in Example 6. This example includes a security token embedded in the returned ID-WSF EPR. Parts of the security token have been omitted due to size.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    <QueryResponse xmlns="&DS2Namespace;">
      <Status code="OK"/>

      <wsa:EndpointReference
            notOnOrAfter="2005-08-15T23:18:56Z" >
        <wsa:Address>http://example.com/pip/bob</wsa:Address>

        <wsa:Metadata>
          <ds:Abstract>
            Bob’s personal profile
          </ds:Abstract>

          <ds:ProviderID>http://example.com/</ds:ProviderID>

          <ds:ServiceType>urn:liberty:id-sis-pp:2003-08</ds:ServiceType>

          <ds:Framework Version="2.0" />

          <ds:SecurityContext>
            <ds:SecurityMechID>urn:liberty:security:2006-08:ClientTLS:SAMLV2</ds:SecurityMechID>
            <ds:SecurityMechID>urn:liberty:security:2005-02:ClientTLS:SAML</ds:SecurityMechID>

            <sec:Token usage="urn:liberty:security:tokenusage:2006-08:SecurityToken" >
              <saml2:Assertion  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                       ID="sxJu9g/vvLG9sAN9bKp/8q0NKU="
                       Issuer="idp.example.com"
                       IssueInstant="2003-09-09T16:58:33.173Z">
                <ds:Signature>...</ds:Signature>
                <saml2:Subject>
                  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
                    http://serviceprovider.com/
                  </saml2:NameId>

                  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
                    <saml2:SubjectConfirmationData xsi:type="saml:KeyInfoConfirmationDataType">
                      <ds:KeyInfo>
                        <ds:KeyName>
                          CN=serviceprovider.com,
                          OU=Services R US,O=Service Nation,...
                        </ds:KeyName>
                      </ds:KeyInfo>
                    </saml2:SubjectConfirmationData>
                  </saml2:SubjectConfirmation>
                </saml2:Subject>

                <saml2:AuthnStatement  AuthnInstant="2003-09-09T16:57:30.000Z"
                            SessionIndex="..."
                            SessionNotOnOrAfter="..."
                            >
                  <saml2:AuthnContext
                   ...
                  </saml2:AuthnContext>
                </saml2:AuthnStatement>

              </saml2:Assertion>
            </sec:Token>
          </ds:SecurityContext>
        </wsa:Metadata>
      </wsa:EndpointReference>
    </QueryResponse>
  </soap:Body>
</soap:Envelope>
```

Example 6. SOAP-bound `<QueryResponse>` Message with Embedded Security Token

3.3.4. DiscoveryQuery Processing Rules

The discovery Service returns entries based on the requester's search criteria (interpreted as described above in Section 3.3.2: `<Query>` Message), the policies of the discovery resource, and the contents of the discovery resource.

For each `<RequestedService>` element in a `<Query>` message, the matching rules MUST applied independently (as if the other `<RequestedService>` elements were not present (potentially returning equivalent EPRs in response to different `<RequestedService>` elements).

When building the results for a given `<RequestedService>` element, the Discovery Service SHOULD return the data in as few EPRs as possible (within the constraints of the EPR) especially with respect to data originating from the same SvcMD.

The Discovery Service SHOULD, when possible, provide the security tokens necessary for the security mechanism(s) identified in the ID-WSF EPRs in the response. If the Discovery Service is not able to generate the necessary security token, it should indicate so by including an empty `<sec:Token>` element with the `ref` attribute set to the value:

```
        urn:liberty:disco:tokenref:ObtainFromIDP
```

The Discovery Service SHOULD mint new EPRs such that they carry the same identity context that was used to invoke the Discovery Service in the invocation context for the targeted WSP.

When minting EPRs in response to a request, the Discovery Service:

- MUST include in the EPR the matching (or all, if none were specified on the request) `<Option>` values contained within the `<ServiceContext>` of the `<SvcMD>` that matched the request criteria.
- SHOULD NOT include any data from the `<SvcMD>` that was not specified in the request **and** is not necessary to create a useful EPR. For example, if the request specified a single `<SecurityMechID>` value, the resulting EPRs should not include other `<SecurityMechID>` elements, even if they are present in the `<SvcMD>` and otherwise could be specified in the same EPR.

The Discovery Service MAY order `<wsa:EndpointReference>` elements as it sees fit. If the Discovery Service is rank ordering the entries, it MUST use descending rank order. This enables the requester to assume that if the results were ordered, the first result is the most relevant.

The following rules specify the status code in the response:

- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.
- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NoResults* as a second-level status code.

  The service may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.4. Operation: *MDAssociationAdd*

The *MDAssociationAdd* operation is used by the WSP to add an association of the principal to the specified metadata.

3.4.1. `wsa:Action` values for `MDAssociationAdd` Messages

`<SvcMDAssociationAdd>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDAssociationAdd`."

`<SvcMDAssociationAddResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDAssociationAddResponse`."

3.4.2. `SvcMDAssociationAdd` Message

The `<SvcMDAssociationAdd>` is called with one or more `<SvcMDID>` elements to add associations to these service metadata descriptions for the principal.

A WSP SHOULD NOT associate the same `<SvcMD>` (or different SvcMD element that carry metadata for the "same" service) to a principal multiple times without first removing the previous entry.

The values in the `<SvcMDID>` element(s) must have been obtained via one of the service metadata operations discussed later in this specification.

```
<!-- SvcMDAssociationAdd operation -->

<xs:element name="SvcMDAssociationAdd" type="SvcMDAssociationAddType"/>

<xs:complexType name="SvcMDAssociationAddType">
  <xs:sequence>
    <xs:element ref="SvcMDID" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 19. `<SvcMDAssociationAdd>` — Schema Fragment

An example message body containing a `<SvcMDAssociationAdd>` message follows. This request adds a new association for the current principal (note that the identity of the principal is carried in the invocation context and not in the body of the message).

```
<ds:SvcMDAssociationAdd>
   <ds:SvcMDID>2323872</ds:SvcMDID>
</ds:SvcMDAssociationAdd>
```

Example 7. `<SvcMDAssociationAdd>` Message

3.4.3. `SvcMDAssociationAddResponse` Message

This response to the `<SvcMDAssociationAdd>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.

```

<!-- Response for SvcMDAssociationAdd operation -->

<xs:element name="SvcMDAssociationAddResponse"
        type="SvcMDAssociationAddResponseType"/>

<xs:complexType name="SvcMDAssociationAddResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

```

Figure 20. `<SvcMDAssociationAddResponse>` — Schema Fragment

```

 <ds:SvcMDAssociationAddResponse>
    <lu:Status code="OK" />
 </ds:SvcMDAssociationAddResponse>

```

Example 8. `<SvcMDAssociationAddResponse>` Message

3.4.4. MDAssociation Add Processing Rules

- Once the association is added by the WSP, the Discovery Service MUST consider this metadata (subject to local policy) when responding to subsequent DiscoveryQuery operations and should the associated metadata meet the requirements of the query, mint the necessary ID-WSF EPRs based upon the requirements of the WSC and the WSP.
- The Discovery Service SHOULD reject attempts to associate a `<SvcMDID>` that has already been associated with the principal by this WSP. In such cases the Discovery service MAY set the second level status code in the response to *Duplicate*.
- The Discovery Service MAY similarly reject attempts to associate a `<SvcMDID>` that references the same service type and WSP that is in one of the already associated service metadata descriptions. In such cases the Discovery service MAY set the second level status code in the response to *LogicalDuplicate*.
- The Discovery Service MUST reject attempts to associate a `<SvcMDID>` that does not exist or is not owned by the WSP invoking the call. In such cases the Discovery service MAY set the second level status code in the response to *NotFound*.
- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.
- If the top-level status code is *Failed*, the response MAY also contain *Forbidden*, *Duplicate*, *LogicalDuplicate*, or *NotFound* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.
- A Discovery Service MAY provide some programmatic or browser based interface which allows the principal to manage the service associations that have been added to their resource at the Discovery Service. A principal may be able to use such interfaces to change or even remove service associations made by the WSP without the WSP's permission (it is the principal's resource) and perhaps, even without notification to the WSP.

  Such interfaces are out-of-scope for this specification, but are mentioned here to remind the WSP that they may exist.

3.5. Operation: *MDAssociationQuery*

The *MDAssociationQuery* operation is used by the WSP to query the Discovery Service for any previously added associations related to the principal.

3.5.1. `wsa:Action` values for `MDAssociationQuery` Messages

`<SvcMDAssociationQuery>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDAssociationQuery`."

`<SvcMDAssociationQueryResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDAssociationQueryResponse`."

3.5.2. `SvcMDAssociationQuery` Message

The `<SvcMDAssociationQuery>` is called with zero or more `<SvcMDID>` elements to query associations to these service metadata descriptions. If no `<SvcMDID>` elements are specified, ALL associations between the WSP's service metadata and the principal are returned.

```

<!-- SvcMDAssociationQuery operation -->

<xs:element name="SvcMDAssociationQuery" type="SvcMDAssociationQueryType"/>

<xs:complexType name="SvcMDAssociationQueryType">
  <xs:sequence>
    <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

```

Figure 21. `<SvcMDAssociationQuery>` — Schema Fragment

An example message body containing a `<SvcMDAssociationQuery>` message follows. This request asks for all associations.

```

 <ds:SvcMDAssociationQuery />

```

Example 9. `<SvcMDAssociationQuery>` Message

3.5.3. `SvcMDAssociationQueryResponse` Message

This response to the `<SvcMDAssociationQuery>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.
- `<SvcMDID>`: the associated service metadata ID(s). If `<SvcMDID>`s were specified on the `<SvcMDAssociationQuery>` the response will be limited to at most those IDs (if they have been associated with the principal).

```
<!-- Response for SvcMDAssociationQuery operation -->

<xs:element name="SvcMDAssociationQueryResponse"
        type="SvcMDAssociationQueryResponseType"/>

<xs:complexType name="SvcMDAssociationQueryResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
    <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 22. `<SvcMDAssociationQueryResponse>` — Schema Fragment

```
<ds:SvcMDAssociationQueryResponse>
   <lu:Status code="OK" />
   <ds:SvcMDID>2323872</ds:SvcMDID>
</ds:SvcMDAssociationQueryResponse>
```

Example 10. `<SvcMDAssociationQueryResponse>` Message

3.5.4. MDAssociation Query Processing Rules

- The Discovery Service MUST limit the operation to only those associations added by the WSP to the current principal's resource (a WSP MUST NOT be able to query associations added at the same Discovery Service by other WSPs or associations added to a different principal). There MUST NOT be any indication on the response as to whether or not other such elements exist.
- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.
- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NotFound* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.6. Operation: *MDAssociationDelete*

The *MDAssociationDelete* operation is used by the WSP to delete a previously added association of the principal to the specified metadata.

3.6.1. `wsa:Action` values for `MDAssociationDelete` Messages

`<SvcMDAssociationDelete>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDAssociationDelete`."

`<SvcMDAssociationDeleteResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDAssociationDeleteResponse`."

3.6.2. `SvcMDAssociationDelete` Message

The `<SvcMDAssociationDelete>` is called with one or more `<SvcMDID>` elements to delete associations to these service metadata descriptions.

Note that the service metadata description is not impacted by this call. Only the principal's association with the metadata is impacted.

```
<!-- SvcMDAssociationDelete operation -->

<xs:element name="SvcMDAssociationDelete" type="SvcMDAssociationDeleteType"/>

<xs:complexType name="SvcMDAssociationDeleteType">
  <xs:sequence>
    <xs:element ref="SvcMDID" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 23. `<SvcMDAssociationDelete>` — Schema Fragment

An example message body containing a `<SvcMDAssociationDelete>` message follows. This request deletes a single association for the current principal (note that the identity of the principal is carried in the invocation context and not in the body of the message).

```
<ds:SvcMDAssociationDelete>
   <ds:SvcMDID>2323872</ds:SvcMDID>
</ds:SvcMDAssociationDelete>
```

Example 11. `<SvcMDAssociationDelete>` Message

3.6.3. `SvcMDAssociationDeleteResponse` Message

This response to the `<SvcMDAssociationDelete>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.

```
<!-- Response for SvcMDAssociationDelete operation -->

<xs:element name="SvcMDAssociationDeleteResponse"
        type="SvcMDAssociationDeleteResponseType"/>

<xs:complexType name="SvcMDAssociationDeleteResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 24. `<SvcMDAssociationDeleteResponse>` — Schema Fragment

```
<ds:SvcMDAssociationDeleteResponse>
   <lu:Status code="OK" />
</ds:SvcMDAssociationDeleteResponse>
```

Example 12. `<SvcMDAssociationDeleteResponse>` Message

3.6.4. MDAssociation Delete Processing Rules

- Once deleted, the association MUST NOT be subsequently used by the DS to mint ID-WSF EPRs in response to queries relative to this principal. However, WSPs should be prepared to receive requests from WSCs from clients who previously obtained ID-WSF EPRs minted from the associaton which haven't expired.

- The Discovery Service MUST limit the operation to only those associations added by the WSP to the current principal's resource (a WSP MUST NOT be able to delete associations added at the same Discovery Service by other WSPs or associations added to a different principal). There MUST NOT be any indication on the response as to whether or not other such elements exist.

- The Discovery Service MUST treat attempts to delete non-existant associations as a successful no-op. This applies whether or not there are other existing associations being deleted in the same request (so a request to delete a single association that doesn't exist will succeed, even though the Discovery Service does not have to actually delete the record).

- This operation MUST be atomic and if successful, all portions of the request MUST have succeeded. If any portion of the request fails, the entire request must fail.

- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.

- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.7. Operation: *MetadataRegister*

The *MetadataRegister* operation is used to register a new service metadata description with the Discovery Service.

3.7.1. `wsa:Action` values for `MetadataRegister` Messages

`<SvcMDRegister>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDRegister`."

`<SvcMDRegisterResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDRegisterResponse`."

3.7.2. `SvcMDRegister` Message

The `<SvcMDRegister>` is called with one or more service metadata descriptions to be registered at the Discovery Service on behalf of the WSP.

```

<!-- Register operation for Service Metadata -->

<xs:element name="SvcMDRegister" type="SvcMDRegisterType"/>

<xs:complexType name="SvcMDRegisterType">
  <xs:sequence>
    <xs:element ref="SvcMD" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


```

Figure 25. `<SvcMDRegister>` — Schema Fragment

An example message body containing a `<SvcMDRegister>` message follows. This request registers a new service metadata description. Note that the WSP has not set the `svcMDID` attribute on the `<SvcMD>` element – this will be assigned by the DS and returned in the response to the WSP.

```
<ds:SvcMDRegister>
   <ds:SvcMD>
      <ds:Abstract>Profile Service</ds:abstract>
      <ds:ProviderID>http://profile.com</ds:ProviderID>
      <ds:ServiceContext>
         <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
         <ds:EndpointContext>
            <ds:Address>https://profile.com/</ds:Address>
            <sb:Framework version="2.0" />
            <ds:SecurityMechID>
               urn:liberty:security:2003-08:TLS:Bearer
            </ds:SecurityMechID>
         </ds:EndpointContext>
      </ds:ServiceContext>
   </ds:SvcMD>
</ds:SvcMDRegister>
```

Example 13. `<SvcMDRegister>` Message

3.7.3. `SvcMDRegisterResponse` Message

This response to the `<SvcMDRegister>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.
- One or more `<SvcMDID>` if the call was successful (status code is OK). One SvcMDID is returned for each service metadata element registered.
- `<Keys>`: Contains the key descriptors for the keys used by the Discovery Service to sign security tokens (see Section 3.12 for a description of when and why this may be necessary).

```
<!-- Response for SvcMDRegister operation -->

<xs:element name="SvcMDRegisterResponse"
         type="SvcMDRegisterResponseType"/>

<xs:complexType name="SvcMDRegisterResponseType">
  <xs:sequence>

    <xs:element ref="lu:Status" />
    <xs:element ref="SvcMDID"  minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="Keys"    minOccurs="0" maxOccurs="unbounded" />

  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 26. `<SvcMDRegisterResponse>` — Schema Fragment

```
<ds:SvcMDRegisterResp>
   <lu:Status code="OK"/>
   <ds:SvcMDID>2323872</ds:SvcMDID>
</ds:SvcMDRegister>
```

Example 14. `<SvcMDRegisterResponse>` Message

3.7.4. Metadata Register Processing Rules

- This operation MUST be processed in the context of the WSP, (as opposed to the context of the principal) so that the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.

  Even if this operation is invoked with an invocation identity of a principal, the Discovery Service MUST use the Sender's identity (the WSP) when processing this call. The Discovery Service MAY refuse to process the operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.

- The transaction unit for this operation is the entire set of `<SvcMD>` elements; they either all succeed or all fail. The Discovery Service MUST enforce this atomicity.

- For each `<SvcMD>` element, the Discovery Service instance MAY store the metadata provided such that it can be used (subject to policy) to mint ID-WSF EPRs in response to future *DiscoveryQuery* operations should that service metadata be associated with a principal's resource at the Discovery Service.

  If the Discovery Service instance does not store the metadata, it MUST return a *Failed* status code for the operation, and therefore not register any of the other entries provided.

  If the Discovery Service does store the metadata, it MUST assign a permanent identifier for the metadata usable by the WSP to subsequently reference the metadata. This identifier MUST be unique across all metadata objects stored by a WSP and MAY be unique across all metadata objects stored by all WSPs at that Discovery Service. This identifier is provided to the WSP in the response and can be subsequently used by the WSP to associate this metadata with a principal or to manage the metadata using one of the other metadata operations.

- A WSP MAY register multiple service metadata descriptions that for all intents and purposes, appear to be fully equal. The Discovery Service MUST NOT generate an error solely because it thinks the descriptions are equal. The Discovery Service MUST treat these records as independent registrations and assign the associated unique SvcMDID values.

- The Discovery Service MAY have some policy driven limit on the number of service metadata descriptions that it will allow a WSP to register. If a WSP attempts to register a new service metadata description that would exceed such a limit, the DS SHOULD include a secondary-level status code of *LimitExceeded*.

  A WSP should exercise care to only register new service metadata descriptions when an existing, registered, description that meets the WSP's needs is not available.

- The Discovery Service SHOULD validate that a given SvcMD only contains entries for a single logical service (i.e., allow for different versions, but **not** allow differences in the basic service type). For example, a single SvcMD SHOULD not contain data for both a contact book service and a calendar service.

  The Discovery Service MAY, subject to local policies, perform additional validations on the content of a SvcMD.

  If any validation on the SvcMD fails, the Discovery Service SHOULD reject the registration request and MAY include a secondary-level status code of *Invalid*.

- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.

- If the top-level status code is *Failed*, the response MAY also contain *Forbidden*, *Invalid*, or *OverLimit* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.8. Operation: *MetadataQuery*

The *MetadataQuery* operation is used to query the Discovery Service for existing, registered, service metadata descriptions.

3.8.1. `wsa:Action` values for `MetadataQuery` Messages

`<SvcMDQuery>` messages MUST set the value of the `<wsa:Action>` header to "`urn:liberty:disco:2006-08:SvcMDQuery`."

`<SvcMDQueryResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDQueryResponse`."

3.8.2. `SvcMDQuery` Message

The `<SvcMDQuery>` is called with zero or more `<SvcMDID>` elements to retrieve the specified list of service metadata descriptions. If no `<SvcMDID>`s are specified, ALL of the metadata stored at the Discovery service by the invoking WSP will be returned.

```

<!-- Query operation on Service Metadata -->

<xs:element name="SvcMDQuery" type="SvcMDQueryType"/>

<xs:complexType name="SvcMDQueryType">
  <xs:sequence>
    <xs:element ref="SvcMDID"
            minOccurs="0"
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


```

Figure 27. `<SvcMDQuery>` — Schema Fragment

An example message body containing a `<SvcMDQuery>` message follows. This request queries for a specific service metadata description by providing the ID of the desired metadata in the `<SvcMDID>` element.

```

 <ds:SvcMDQuery>
    <ds:SvcMDID>2323872</ds:SvcMDID>
 </ds:SvcMDQuery>

```

Example 15. `<SvcMDQuery>` Message

3.8.3. `SvcMDQueryResponse` Message

This response to the `<SvcMDQuery>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.
- One or more `<SvcMD>` elements if the call was successful (status code is OK).

```

<!-- Response for Query operation on Service Metadata -->

<xs:element name="SvcMDQueryResponse" type="SvcMDQueryResponseType"/>

<xs:complexType name="SvcMDQueryResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
    <xs:element ref="SvcMD" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


```

Figure 28. `<SvcMDQueryResponse>` — Schema Fragment

```

 <ds:SvcMDQueryResponse>
    <lu:Status code="OK" />
    <ds:SvcMD svcMDID="2323872" >
       <ds:Abstract>Profile Service</ds:Abstract>
       <ds:ProviderID>http://profile.com</ds:ProviderID>
       <ds:ServiceContext>
          <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
          <ds:EndpointContext>
             <ds:Address>https://profile.com/</ds:Address>
             <sb:Framework version="2.0" />
             <ds:SecurityMechID>
                urn:liberty:security:2003-08:TLS:Bearer
             </ds:SecurityMechID>
          </ds:EndpointContext>
       </ds:ServiceContext>
    </ds:SvcMD>
 </ds:SvcMDQueryResponse>

```

Example 16. `<SvcMDQueryResponse>` Message

3.8.4. Metadata Query Processing Rules

- This operation MUST be processed in the context of the WSP, (as opposed to the context of the principal) so that the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.

  Even if this operation is invoked with an invocation identity of a principal, the Discovery Service MUST use the Sender's identity (the WSP) when processing this call. The Discovery Service MAY refuse to process the operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.

- The Discovery Service MUST limit the results to only those metadata elements stored by the WSP (a WSP MUST NOT be able to retrieve metadata elements stored at the same Discovery Service by other WSPs). There MUST NOT be any indication on the response as to whether or not other such elements exist.

- The Discovery Service SHOULD treat a request that matches a subset of the `svcMDID` values specified in the request as a successful request returning the entries that were found and nothing for the missing entries. The WSC will be able to distinguish which entries were found by examining the `svcMDID` attribute on the `<SvcMD>` element(s) in the response.

- If request processing succeeded AND results are returned, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.

- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* or *NoResults* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.9. Operation: *MetadataReplace*

The *MetadataReplace* operation is used by a WSP to replace previously stored metadata in the Discovery Service. This is how the WSP updates their metadata without having to reassociate with a principal.

3.9.1. `wsa:Action` values for `MetadataReplace` Messages

`<SvcMDReplace>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDReplace`."

`<SvcMDReplaceResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDReplaceResponse`."

3.9.2. `SvcMDReplace` Message

The `<SvcMDReplace>` is called with one or more replacement `<SvcMD>` elements each of which must include the `svcMDID` attribute set to the ID of the respective metadata element they are to replace.

```

<!-- Replace operation on Service Metadata -->

<xs:element name="SvcMDReplace" type="SvcMDReplaceType"/>

<xs:complexType name="SvcMDReplaceType">
  <xs:sequence>
    <xs:element ref="SvcMD" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


```

Figure 29. `<SvcMDReplace>` — Schema Fragment

An example message body containing a `<SvcMDReplace>` message follows. This request replaces an existing metadata element to update the endpoint for the service.

```

<ds:SvcMDReplace>
   <ds:SvcMD svcMDID="2323872" >
      <ds:Abstract>Profile Service</ds:abstract>
      <ds:ProviderID>http://profile.com</ds:ProviderID>
      <ds:ServiceContext>
         <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
         <ds:EndpointContext>
            <ds:Address>https://newaddr.com/</ds:Address>
            <sb:Framework version="2.0"/>
            <ds:SecurityMechID>
               urn:liberty:security:2003-08:TLS:Bearer
            </ds:SecurityMechID>
         </ds:EndpointContext>
      </ds:ServiceContext>
   </ds:SvcMD>
</ds:SvcMDReplace>

```

Example 17. `<SvcMDReplace>` Message

3.9.3. `SvcMDReplaceResponse` Message

This response to the `<SvcMDReplace>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.

```
<!-- Response for SvcMDReplace operation -->

<xs:element name="SvcMDReplaceResponse" type="SvcMDReplaceResponseType"/>

<xs:complexType name="SvcMDReplaceResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 30. `<SvcMDReplaceResponse>` — Schema Fragment

```
<ds:SvcMDReplaceResponse>
   <lu:Status code="OK"/>
</ds:SvcMDReplaceResponse>
```

Example 18. `<SvcMDReplaceResponse>` Message

3.9.4. Metadata Replace Processing Rules

- This operation MUST be processed in the context of the WSP, (as opposed to the context of the principal) so that the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.

  Even if this operation is invoked with an invocation identity of a principal, the Discovery Service MUST use the Sender's identity (the WSP) when processing this call. The Discovery Service MAY refuse to process the operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.

- The Discovery Service MUST limit the operation to only those metadata elements stored by the WSP (a WSP MUST NOT be able to replace metadata elements stored at the same Discovery Service by other WSPs). There MUST NOT be any indication on the response as to whether or not other such elements exist.

- The Discovery Service SHOULD validate that the replacement SvcMD contains the same logical service as the original SvcMD. By "logical service" we mean to allow for different versions, but **not** allow differences in the basic service type. For example, a calendar service SvcMD SHOULD not be allowed to replace a contact book service SvcMD.

  The Discovery Service SHOULD also validate that the replacement SvcMD only contains entries for a single logical service (as described above). For example, a single SvcMD SHOULD not contain data for both a contact book service and a calendar service.

  The Discovery Service MAY, subject to local policies, perform additional validations on the content of a SvcMD.

  If any validation on the SvcMD fails, the Discovery Service SHOULD reject the replacement request and MAY include a secondary-level status code of *Invalid*.

- The transaction unit for this operation is the entire set of `<SvcMD>` elements; they either all succeed or all fail. The Discovery Service MUST enforce this atomicity.
- Once replaced, the previous service metadata element MUST NOT be subsequently used by the DS to mint ID-WSF EPRs. However, WSPs should be prepared to receive requests from WSCs from clients who previously obtained ID-WSF EPRs minted from the prior metadata which haven't expired.
- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.
- If the top-level status code is *Failed*, the response MAY also contain *Forbidden*, *Invalid*, or *NotFound* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.10. Operation: *MetadataDelete*

The *MetadataDelete* operation is used by the WSP to delete previously registered metadata elements in the Discovery Service.

3.10.1. `wsa:Action` values for `MetadataDelete` Messages

`<SvcMDDelete>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDDelete`."

`<SvcMDDeleteResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:disco:2006-08:SvcMDDeleteResponse`."

3.10.2. `SvcMDDelete` Message

The `<SvcMDDelete>` is called with one or more `<SvcMDID>` elements to delete the specified list of service metadata descriptions.

```

<!-- Delete operation on Service Metadata -->

<xs:element name="SvcMDDelete" type="SvcMDDeleteType"/>

<xs:complexType name="SvcMDDeleteType">
  <xs:sequence>
    <xs:element ref="SvcMDID" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


```

Figure 31. `<SvcMDDelete>` — Schema Fragment

An example message body containing a `<SvcMDDelete>` message follows. This request deletes a single service metadata description.

```

<ds:SvcMDDelete>
   <ds:SvcMDID>2323872</ds:SvcMDID>
</ds:SvcMDDelete>

```

Example 19. `<SvcMDDelete>` Message

3.10.3. `SvcMDDeleteResponse` Message

This response to the `<SvcMDDelete>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.

```

<!-- Response for delete operation on Service Metadata -->

<xs:element name="SvcMDDeleteResponse" type="SvcMDDeleteResponseType"/>

<xs:complexType name="SvcMDDeleteResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


```

Figure 32. `<SvcMDDeleteResponse>` — Schema Fragment

```

 <ds:SvcMDDeleteResponse>
    <lu:Status code="OK"/>
 </ds:SvcMDDeleteResponse>

```

Example 20. `<SvcMDDeleteResponse>` Message

3.10.4. Metadata Delete Processing Rules

- This operation MUST be processed in the context of the WSP, (as opposed to the context of the principal) so that the WSP can maintain a single set of service metadata across all principals at the same Discovery Service.

  Even if this operation is invoked with an invocation identity of a principal, the Discovery Service MUST use the Sender's identity (the WSP) when processing this call. The Discovery Service MAY refuse to process the operation if the identity of the Sender cannot be established to the Discovery Service's satisfaction.

- If the service metadata being deleted is still associated with one or more principals, the Discovery Service MUST automatically remove such associations (i.e., the delete of metadata cascades to delete the associations).

- Once deleted, the service metadata element MUST NOT be subsequently used by the DS to mint ID-WSF EPRs. However, WSPs should be prepared to receive requests from WSCs from clients who previously obtained ID-WSF EPRs minted from the metadata which haven't expired.

- The Discovery Service MUST limit the operation to only those metadata elements stored by the WSP (a WSP MUST NOT be able to delete metadata elements stored at the same Discovery Service by other WSPs). There MUST NOT be any indication on the response as to whether or not other such elements exist.

- The Discovery Service MUST treat attempts to delete non-existant metadata elements as a successful no-op. This applies whether or not there are other existing metadata elements being deleted in the same request (so a request to delete a single metadata element that doesn't exist will succeed, even though the Discovery Service does not have to actually delete the record).

- This operation MUST be atomic and if successful, all portions of the request MUST have succeeded. If any portion of the request fails, the entire request must fail.

- If request processing succeeded, the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.
- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* as a second-level status code. The Discovery Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.11. `Option` Value for Response Authentication

The ID-WSF EPR `<SecurityContext>` element provides a way for services to indicate to clients what mechanisms are necessary for the client to authenticate itself to the service via the `<SecurityMechID>` element. The `<SecurityMechID>` values defined by [LibertySecMech] also indicate whether the service uses peer entity authentication (for example, server-side SSL/TLS). However, a web service client may need to know whether the service will use message authentication (that is, whether the service will sign the response message) and may not be willing to use a service which does not sign its responses.

To avoid situations where a client requests data and then discovers it does not trust it because it is not signed, an `<Option>` value is defined:

urn:liberty:disco:2006-08:options:security-response-x509

If a service instance always authenticates its response messages according to the "X.509 v3 Certificate Message Authentication" mechanism in [LibertySecMech], registrations of ID-WSF EPRs describing the service instance SHOULD include this option value. Otherwise, its registered ID-WSF EPRs MUST NOT include this option value. Clients MAY include this option value in `<Query>` messages in order to locate only services which always authenticate their response messages. A service MAY authenticate its response messages even if this option value was not included in its description at the Discovery Service instance.

In case the service also supports a previous version of the security mechanism specification [LibertySecMech11], it should be able to register two different endpoints at the Discovery Service, each of them with different Options values—one according to [LibertySecMech], the other one according to [LibertySecMech11]. This information will aid the client in determining which version of the WSS-SMS specification ([wss-sms-draft] and/or [wss-sms]) is supported by the service, and the service will act accordingly, depending on the ID-WSF EPR used by the client. Note that this behavior only applies to the case when the client's request does not use message authentication mechanisms.

Otherwise, it should be possible for the service to determine the version of the WSS-SMS specification supported by the client by simply analyzing the `<wsse:Security>` header present in the request.

In general, it is recommended that services do not sign their responses unless they positively know that clients are able to perform message authentication and are aware of the version of the WSS-SMS spec used by that client.

3.12. Including Keys in the `SvcMDRegisterResponse` Message

The Discovery Service instance may need to generate signed security tokens in `<QueryResponse>` messages for the ID-WSF EPRs in question (which are later included in a message to a WSP). The WSP which receives the signed security tokens from a client needs to be able to verify the Discovery service instance's signature on the security tokens. Typically the metadata (see [SAMLMeta2]) for the Discovery service instance is sufficient for such information. In certain situations, such as when the Discovery service instance is hosted on a LUAD (see [LibertyClientProfiles]), it may not be feasible to assign the LUAD a ProviderID with which to obtain metadata. However, the key material still needs to be made available to service instances which register ID-WSF EPRs with the Discovery Service which include security mechanisms requiring such tokens.

The Discovery Service instance may include a `<Keys>` element in the `<SvcMDRegisterResponse>` in order to provide such keys.

The Discovery Service instance SHOULD ONLY include the `<Keys>` element in `<SvcMDRegisterResponse>` messages if it has no `<ProviderID>` and the `<SvcMDRegister>` message included Service Metadata that relies upon signed security tokens for one or more of its security mechanisms.

```

  <!-- Keys Element - For use in ModifyResponse -->

  <xs:element name="Keys" type="KeysType"/>

  <xs:complexType name="KeysType">
    <xs:sequence>
      <xs:element ref="md:KeyDescriptor"
              minOccurs="1"
              maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>


```

Figure 33. `<Keys>` — Schema Fragment

The `<Keys>` element appears as a child of the `<SvcMDRegisterResponse>` element. It contains one or more `<KeyDescriptor>` elements.

3.13. Discovery Service Example Messages (NON-Normative)

This section walks through a series of messages to show examples of inputs and outputs. The information in this seciton is **NOT** normative with respect to the specification (in cases where the other normative section(s) of the specification conflict with this section, the normative section(s) will prevail).

Notes about this sequence:

- It is an actual network capture of a real session between two independent liberty implementations.
- The messages are from a **test sequence** which exercises the features of the Discovery Service. We do **not** expect that any real world situation would result in this sequence of messages (or anything similar to this sequence of messages).
- All of these messages were invoked using the same SAML Assertion to establish the invocation context with the principal as the subject and the WSP in the subject confirmation. Note that some of the processing rules for the Discovery Service interfaces (those that manage the service metadata) require that such an invocation context be interpreted as a WSP invoker invocation context.
- The initial state is that the principal has a single service associated with their identity (an instance of the ID-WSF People Service).
- The messages **are** part of a sequence that was invoked in this order and one request (such as a new registration) can and usually does impact the results of subsequent requests.
- The first request and response messages are shown as a full SOAP-bound ID-* message (with all of the SOAP headers). The remaining request and response messages are shown with just the ID-* message component (i.e., only the contents within the `<S:Body>` element are shown).
- The messages have been formated for easier reading (pretty-printing) and data has been ellided (such as the contents of SAML assertions) for brevity.
- Many of the service types used in these example messages do **not** exist as real ID-* services and are only used here as testing input to exercise the Discovery Service.

3.13.1. Query People Service

Query for the People Service

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsse="http://.../oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sb2="urn:liberty:sb:2006-08"
    xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:MessageID S:mustUnderstand="true"
       S:actor="http://schemas.xmlsoap.org/soap/actor/next" id="msgHdrID">
      uuid:asdqwer-238asf-44353608-000b8c14
    </wsa:MessageID>
    <wsa:To     S:mustUnderstand="true"
       S:actor="http://schemas.xmlsoap.org/soap/actor/next" id="wsaToID">
      https://s-ds.liberty-iop.org:8681/DISCO-S
    </wsa:To>
    <wsa:Action   S:mustUnderstand="true"
       S:actor="http://schemas.xmlsoap.org/soap/actor/next" id="wsaActionID">
      urn:liberty:disco:2006-08:Query
    </wsa:Action>
    <wsse:Security S:mustUnderstand="true"
       S:actor="http://schemas.xmlsoap.org/soap/actor/next">
      <sa:Assertion
         xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
         ID="CRED6q6HqDRRCAPsq3L8d_sh"
         IssueInstant="2006-04-06T15:39:12Z"
         Version="2.0">
        ... assertion data was here ...
      </sa:Assertion>
      <wsu:Timestamp wsu:Id="WsuTimestampID">
        <wsu:Created>2006-04-06T15:38:48Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>
  </S:Header>
  <S:Body wsu:id="msgBodyID">
    <disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
      <disco:RequestedService>
        <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
      </disco:RequestedService>
    </disco:Query>
  </S:Body>
</S:Envelope>
```

Things to note about this query:

- it is a query of a service type (in this case, a particular version of the ID-WSF People Service)
- the client has stated that they can support 2 specific security mechanisms (TLS:SAMLV2 and TLS:Bearer) for communicating with the specified service

The response from the Discovery Service:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:lib="urn:liberty:iff:2003-08">
  <soap:Header>
    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing" id="MID">
      uuid:IKvqaaaE0dZ5Hhg1IQZl
    </wsa:MessageID>
    <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
      uuid:asdqwer-238asf-44353608-000b8c14
    </wsa:RelatesTo>
    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
      urn:liberty:disco:2006-08:QueryResponse
    </wsa:Action>
    <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://www.w3.org/2005/03/addressing/role/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsse:Security xmlns:wsse="http://.../oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsu:Timestamp xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsu:Created>2006-04-06T15:39:13Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>
    <sb2:Sender xmlns:sb2="urn:liberty:sb:2006-08"
        xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
        providerID="https://s-ds.liberty-iop.org:8681/idp.xml"
        soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
        wsu:Id="PRV"/>
    <sb2:Framework
        xmlns:sb2="urn:liberty:sb:2006-08"
        version="2.0"
        soap:actor="http://schemas.xmlsoap.org/soap/actor/next"/>
  </soap:Header>
  <soap:Body>
    <disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
      <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
      <wsa:EndpointReference
          xmlns:wsa="http://www.w3.org/2005/08/addressing"
          xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
          notOnOrAfter="2006-04-06T17:39:14Z"
          wsu:Id="EPRIDIo6l485WDEA70lqHi4Ey">
        <wsa:Address>https://s-wsp.liberty-iop.org:8743/PS-PSBEARER</wsa:Address>
        <wsa:Metadata>
          <disco:Abstract>SYMfiam urn:liberty:ps:2006-01 service</disco:Abstract>
          <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
          <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
          <disco:ServiceType>urn:liberty:ps:2006-01</disco:ServiceType>
          <disco:SecurityContext>
            <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
            <sec:Token xmlns:sec="urn:liberty:security:2006-08"
                usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
              <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
                  ID="CREDI4cINqS4SV1HPm_xOoGh"
                  IssueInstant="2006-04-06T15:39:14Z"
                  Version="2.0">
                ... assertion data was here ...
              </sa:Assertion>
            </sec:Token>
          </disco:SecurityContext>
        </wsa:Metadata>
      </wsa:EndpointReference>
    </disco:QueryResponse>
  </soap:Body>
</soap:Envelope>
```

Things to note about this response:

- the query was successful (status code is OK).
- there is a single ID-WSF EPR in the response for the service type specified in the request
- the details within the assertion were edited out to save space

3.13.2. Query provider

Query for the same service using the ProviderID of the WSP.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- it is a query of a ProviderID, so all services provided by that ProviderID would be returned (and depending upon the invocation context of the ID-WSF framework, this may be all services provided by that provider for a particular principal or just all services).
- the client has stated that they can support 2 specific security mechanisms (TLS:SAMLV2 and TLS:Bearer) for comminicating with the specified service

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference
       xmlns:wsa="http://www.w3.org/2005/08/addressing"
       xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
       notOnOrAfter="2006-04-06T17:39:18Z"
       wsu:Id="EPRIDyGxcYpQ8Gace5y8lDm7Z">
    <wsa:Address>https://s-wsp.liberty-iop.org:8743/PS-PSBEARER</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>SYMfiam urn:liberty:ps:2006-01 service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:liberty:ps:2006-01</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion
             xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
             ID="CREDOEV7U7-mJk02pEVJAn--"
             IssueInstant="2006-04-06T15:39:18Z"
             Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- there is a single ID-WSF EPR in the response denoting the single service offered by the provider specified in the request for this user.
- The ID-WSF EPR is **almost** identical to the ID-WSF EPR returned in the previous request. The differences are only in the timestamps and the element IDs.

3.13.3. Query (empty)

A Query without specifying any search criteria (which should return all services available to the user).

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq"/>
```

Things to note about this query:

- you still need to include the `<disco:Query>` element in the request, just that its contents are empty.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference
       xmlns:wsa="http://www.w3.org/2005/08/addressing"
       xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
       notOnOrAfter="2006-04-06T17:39:21Z" wsu:Id="EPRIDt3MOElDHPL0EBD8whEvw">
    <wsa:Address>https://s-ds.liberty-iop.org:8681/DISCO-S</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>SYMfiam Discovery Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-ds.liberty-iop.org:8681/idp.xml</disco:ProviderID>
      <disco:ServiceType>urn:liberty:disco:2006-08</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
             usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion
               xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
               ID="CREDr2tN6rxYICAXxD6CtenC"
               IssueInstant="2006-04-06T15:39:21Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference
       xmlns:wsa="http://www.w3.org/2005/08/addressing"
       xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
       notOnOrAfter="2006-04-06T17:39:21Z" wsu:Id="EPRIDQBqOfWHDp3GKFSqqnZyj">
    <wsa:Address>https://s-wsp.liberty-iop.org:8743/PS-PSBEARER</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>SYMfiam urn:liberty:ps:2006-01 service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:liberty:ps:2006-01</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
             usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion
               xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
               ID="CREDAsw9RdinqIu3m4HtUxX5"
               IssueInstant="2006-04-06T15:39:22Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- two ID-WSF EPRs were returned, one for the Discovery Service and one for the People Service.
- Discovery Service instances will usually expose ID-WSF EPRs which point to themselves in this way in order to expose alternative invocation methods and/or allow the client to obtain newer credentials.

3.13.4. Query People Service and Provider

A discovery query specifying both the Provider ID and the Service Type (so that only services of that type by that provider are returned)

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ServiceType>urn:liberty:ps:2006-01</disco:ServiceType>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- both the `<disco:ServiceType>` element and the `<disco:ProviderID>` element are included.
- The provider and service type are the same ones we've been been using so we should get the same response we had on earlier requests.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:39:26Z" wsu:Id="EPRIDwrdfxWpBRhCXsEMW1cjo">
    <wsa:Address>https://s-wsp.liberty-iop.org:8743/PS-PSBEARER</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>SYMfiam urn:liberty:ps:2006-01 service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:liberty:ps:2006-01</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
           usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion
              xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDZbnmfBjqRINmQeWvyxCG"
              IssueInstant="2006-04-06T15:39:26Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).

- The ID-WSF EPR is **almost** exactly the same as the ID-WSF EPR returned in the first 2 examples above.

  The differences are only in the timestamps and the element IDs. This is because both EPRs are for the same principal accessing the same service (we just discovered the EPR through a different query).

3.13.5. SvcMDQuery (empty)

A query for all SvcMD stored at the DS on behalf of the invoking provider.

```
<disco:SvcMDQuery xmlns:disco="urn:liberty:disco:2006-08"/>
```

Things to note about this query:

- it's empty – meaning that all registered SvcMDs are requested.
- the query is executed in the context of the provider invoking the query (identified in the invocation context specified in the ID-WSF framework headers)

Server Response:

```
 <disco:SvcMDQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMD svcMDID="SVCMDIDy_7yFAbwuFPkgYIjKCjv">
    <disco:Abstract>SYMfiam urn:liberty:ps:2006-01 service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:liberty:ps:2006-01</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://s-wsp.liberty-iop.org:8743/PS-PSBEARER</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDQueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- one SvcMD was returned for the ID-WSF People Service (which is the only SvcMD registered by the invoking provider).

3.13.6. SvcMDQuery w/Bad SvcMDID

A service metadata query using an invalid SvcMDID.

```
<disco:SvcMDQuery xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>123</disco:SvcMDID>
</disco:SvcMDQuery>
```

Things to note about this query:

- 123 is an invalid SvcMDID

Server Response:

```
<disco:SvcMDQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="Failed">
    <lu:Status code="NoResults"/>
  </lu:Status>
</disco:SvcMDQueryResponse>
```

Things to note about this response:

- the query failed (status code is Failed).
- The optional sub-status is included which indicates that there were *NoResults* for the query.

3.13.7. SvcMDRegister w/single SvcMD

Registration of a single service metadata instance in the DS.

```
<disco:SvcMDRegister xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMD>
    <disco:Abstract>TestDisco Test Payment Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:pmt:2007-11</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://payment.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDRegister>
```

Things to note about this request:

- a single SvcMD is registered
- no `svcMDID` attribute is specified on the SvcMD during registration (it will be assigned by the DS if the request is successful).
- the service being registered is a test payment service provided by the same provider we've seen earlier, supports a single service and framework version and exposes three different security mechanisms.

Server Response:

```
<disco:SvcMDRegisterResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMDID>SVCMDIDg9WP0thd_HvPd427KY9M</disco:SvcMDID>
</disco:SvcMDRegisterResponse>
```

Things to note about this response:

- the registration was successful (status code is OK).
- The SvcMD was assigned the svcMDID *SVCMDIDg9WP0thd_HvPd427KY9M*.

3.13.8. SvcMDQuery w/Good SvcMDID

Query the SvcMD that we just registed using the SvcMDID that was returned in the response.

```
<disco:SvcMDQuery xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDg9WP0thd_HvPd427KY9M</disco:SvcMDID>
</disco:SvcMDQuery>
```

Things to note about this query:

- the SvcMDID is the ID that was returned in the response to the `<SvcMDRegister>` we executed in the previous step.

Server Response:

```
<disco:SvcMDQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMD svcMDID="SVCMDIDg9WP0thd_HvPd427KY9M">
    <disco:Abstract>TestDisco Test Payment Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:pmt:2007-11</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://payment.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDQueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- The SvcMD has the `svcMDID` attribute assigned by the Discovery Service.
- The remaining data is identitical to that registered by the provider in the previous request.

3.13.9. SvcMDDelete w/Good SvcMDID

Delete the Service Metadata that we just registered and queried.

```
<disco:SvcMDDelete xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDg9WP0thd_HvPd427KY9M</disco:SvcMDID>
</disco:SvcMDDelete>
```

Things to note about this query:

- the SvcMDID that we obtained in the registration above is used

Server Response:

```
<disco:SvcMDDeleteResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDDeleteResponse>
```

Things to note about this response:

- the delete was successful (status code is OK).

3.13.10. SvcMDDelete w/Already Deleted SvcMDID

Delete the same service metadata that we just deleted.

```
<disco:SvcMDDelete xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDg9WP0thd_HvPd427KY9M</disco:SvcMDID>
</disco:SvcMDDelete>
```

Things to note about this query:

- the SvcMDID that we already deleted is specified

Server Response:

```
<disco:SvcMDDeleteResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDDeleteResponse>
```

Things to note about this response:

- the delete was successful (status code is OK), even though the SvcMD was already deleted as the delete of a non-existant SvcMD is defined by this specification to be successful.

3.13.11. SvcMDRegister w/Complex SvcMD

A basic registration of a SvcMD with the Discovery Service. This is a little bit different in that the SvcMD is rather complex (which is good for testing the Query interface).

```
<disco:SvcMDRegister xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMD>
    <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
      <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://old-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.1"/>
        <disco:SecurityMechID>urn:liberty:security:2003-08:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2003-08:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>https://old2-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>http://old-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:null:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:null:SAML2</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://calendars.testing.com</disco:Address>
        <disco:Address>https://calendars.testing.backup.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
      <disco:Address>http://calendars.testing.com</disco:Address>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:SecurityMechID>urn:liberty:security:2005-02:null:SAML2</disco:SecurityMechID>
    </disco:EndpointContext>
   </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDRegister>
```

Things to note about this query:

- it's a rather complex SvcMD, but registration is still the same.

Server Response:

```
<disco:SvcMDRegisterResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMDID>SVCMDIDmifyjzIKO6tNd8evymnL</disco:SvcMDID>
</disco:SvcMDRegisterResponse>
```

Things to note about this response:

- the registration was successful (status code is OK).
- The SvcMD was assigned the svcMDID *SVCMDIDmifyjzIKO6tNd8evymnL*.
- Now the fun begins as this SvcMD let's us exercise many of the interesting portions of the Discovery Query interface.

3.13.12. SvcMDQuery w/Good SvcMDID for Complex SvcMD

Query the SvcMD that we just registed using the SvcMDID that was returned in the response to make sure it was registered correctly.

```
<disco:SvcMDQuery xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDmifyjzIKO6tNd8evymnL</disco:SvcMDID>
</disco:SvcMDQuery>
```

Things to note about this query:

- the SvcMDID is the ID that was returned in the response to the `<SvcMDRegister>` we executed in the previous step.

Server Response:

```
<disco:SvcMDQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMD svcMDID="SVCMDIDmifyjzIKO6tNd8evymnL">
    <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
      <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://old-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.1"/>
        <disco:SecurityMechID>urn:liberty:security:2003-08:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2003-08:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>https://old2-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>http://old-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:null:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:null:SAML2</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://calendars.testing.com</disco:Address>
        <disco:Address>https://calendars.testing.backup.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>http://calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:null:SAML2</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDQueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- The SvcMD has the `svcMDID` attribute assigned by the Discovery Service.
- The remaining data is identitical to that registered by the provider in the previous request. Key here is that the ordering and grouping of elements has been maintained.

3.13.13. SvcMDReplace of existing SvcMD

Replace the complex SvcMD that we just registered with a simpler version

```
<disco:SvcMDReplace xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMD svcMDID="SVCMDIDmifyjzIKO6tNd8evymnL">
    <disco:Abstract>TestDisco Test Payment Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
```

```
      <disco:ServiceType>urn:x-test:pmt:2007-11</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://payment.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDReplace>
```

Things to note about this request:

- the svcMDID attribute has the value obtained during the previous registration so that registration should be replaced.

Server Response:

```
<disco:SvcMDReplaceResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDReplaceResponse>
```

Things to note about this response:

- the replacement was successful (status code is OK).
- no other data is returned. The SvcMDID does not change when the SvcMD is replaced.

3.13.14. SvcMDQuery of Replaced SvcMDID

Query the SvcMD that we just replaced.

```
<disco:SvcMDQuery xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDmifyjzIKO6tNd8evymnL</disco:SvcMDID>
</disco:SvcMDQuery>
```

Things to note about this query:

- We use the same svcMDID that we had for the previous SvcMD as the value doesn't change when we do a replacement.

Server Response:

```
<disco:SvcMDQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMD svcMDID="SVCMDIDmifyjzIKO6tNd8evymnL">
    <disco:Abstract>TestDisco Test Payment Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:pmt:2007-11</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://payment.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDQueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- the new SvcMD is returned.

3.13.15. SvcMDDelete of replaced SvcMD

Delete the Service Metadata that we just replaced (to clean up after ourselves).

```
<disco:SvcMDDelete xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDmifyjzIKO6tNd8evymnL</disco:SvcMDID>
</disco:SvcMDDelete>
```

Things to note about this query:

- the SvcMDID that we used in the replacement above is used

Server Response:

```
<disco:SvcMDDeleteResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDDeleteResponse>
```

Things to note about this response:

- the delete was successful (status code is OK).

3.13.16. SvcMDRegister w/multiple SvcMDs

A registration of 3 different SvcMD elements of varying levels of complexity

```
<disco:SvcMDRegister xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMD>
    <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
      <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://1-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.1"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>https://2-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>http://3-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2005-02:null:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://4-calendars.testing.com</disco:Address>
        <disco:Address>https://5-calendars.testing.backup.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>http://6-calendars.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
  <disco:SvcMD>
    <disco:Abstract>TestDisco Test Payment Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:pmt:2007-11</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://payment.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
  <disco:SvcMD>
    <disco:Abstract>TestDisco Test ATM Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
      <disco:Options>
        <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt3</disco:Option>
      </disco:Options>
      <disco:EndpointContext>
        <disco:Address>https://test2.atm.CA.US.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <disco:Action>urn:x-test:atm:2007-11:GetBalance</disco:Action>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>https://readers.atm.CA.US.testing.com</disco:Address>
        <disco:Address>https://readers.atm.NY.US.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <disco:Action>urn:x-test:atm:2007-11:GetBalance</disco:Action>
        <disco:Action>urn:x-test:atm:2007-11:ListAccounts</disco:Action>
      </disco:EndpointContext>
      <disco:EndpointContext>
        <disco:Address>https://writers.atm.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <disco:Action>urn:x-test:atm:2007-11:Withdraw</disco:Action>
        <disco:Action>urn:x-test:atm:2007-11:Transfer</disco:Action>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDRegister>
```

Things to note about this registration:

- there SvcMDs use many of the features of the data structures to define various contexts in which the service can be reached and which actions and/or options are available at said services.
- We do **NOT** represent that the SvcMDs registered here are typical or normal. They were explicitly created to examine/test some of the intricacies of handling queries against the SvcMD data structure.

Server Response:

```
<disco:SvcMDRegisterResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMDID>SVCMDIDMfPMb1wcRSiwnu8D3BGO</disco:SvcMDID>
  <disco:SvcMDID>SVCMDIDeZQGkXuw75O_uh3Q9OLO</disco:SvcMDID>
  <disco:SvcMDID>SVCMDIDrpG8SJpeUdSmla_ZSFUN</disco:SvcMDID>
</disco:SvcMDRegisterResponse>
```

Things to note about this response:

- the registration was successful (status code is OK).
- the `svcMDID` for each of the added elements is returned in sequence. The first `SvcMDID` to the first `<SvcMD>` in the request, the second to the second, and so forth.

3.13.17. Query Calendar Service

Query for the test calendar service for this user.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- the service type on this query is one of those specified in one of the SvcMDs that were just registered in the previous call.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="Failed">
    <lu:Status code="NoResults"/>
  </lu:Status>
</disco:QueryResponse>
```

Things to note about this response:

- the query failed (status code was Failed). This is because while the SvcMD has been *registered*, it has **not** been *associated* with the user.

3.13.18. SvcMDAssociationAdd the Calendar Service SvcMD

Associate a single SvcMD with the current principal.

```
<disco:SvcMDAssociationAdd xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDMfPMb1wcRSiwnu8D3BGO</disco:SvcMDID>
</disco:SvcMDAssociationAdd>
```

Things to note about this query:

- the SvcMDID specified is the SvcMDID assigned to the test calendar service registered above (the first SvcMD in the multi-SvcMD registration).

Server Response:

```
<disco:SvcMDAssociationAddResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDAssociationAddResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- this service is now associated with the principal and available to subsequent Discovery Service queries.

3.13.19. SvcMDAssociationQuery w/SvcMDID

Query the SvcMD Associations to see if that SvcMD is now associated with the principal

```
<disco:SvcMDAssociationQuery xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDMfPMb1wcRSiwnu8D3BGO</disco:SvcMDID>
</disco:SvcMDAssociationQuery>
```

Things to note about this query:

- the SvcMDID provided is the SvcMDID that was just associated with the user in the previous request

Server Response:

```
<disco:SvcMDAssociationQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMDID>SVCMDIDMfPMb1wcRSiwnu8D3BGO</disco:SvcMDID>
</disco:SvcMDAssociationQueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- the matching SvcMDIDs were returned (since you can query more than one and not all may have matched on a successful query)

3.13.20. SvcMDAssociationQuery w/o SvcMDID

Query all SvcMD Associations for the current principal.

```
<disco:SvcMDAssociationQuery xmlns:disco="urn:liberty:disco:2006-08"/>
```

Things to note about this query:

- No SvcMDIDs are specified which causes all SvcMD associations that were created by the invoking provider to be listed.

Server Response:

```
<disco:SvcMDAssociationQueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <disco:SvcMDID>SVCMDIDy_7yFAbwuFPkgYIjKCjv</disco:SvcMDID>
  <disco:SvcMDID>SVCMDIDMfPMb1wcRSiwnu8D3BGO</disco:SvcMDID>
</disco:SvcMDAssociationQueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).

- two SvcMDIDs were returned. One for the service recently associated and one for the ID-WSF People Service that had been previously associated. (This call, as well as the previous registration and association calls, was made in the context of that provider so both should be visible).

3.13.21. Query Calendar Service (again)

Query for the test calendar service for this user (which previously failed, but now the SvcMD has been associated with this principal).

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- again we're looking for the test calendar service (which has now been associated with the principal).
- This query was constructed to be resolvable only by the data within the 2nd `<ServiceContext>` element (it's the only one with the ServiceType "*urn:x-test:cal:2008-03*") and within there, the 2nd `<EndpointContext>` element (it's the only one with the SecurithMechID *...:null:SAMLV2*).

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:40:28Z"
      wsu:Id="EPRID3jLhZ6fsjx3xFNF22Hyx">
    <wsa:Address>http://6-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CRED-HgB2SRBPPtovTK7ckof"
              IssueInstant="2006-04-06T15:40:28Z"
              Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successfull (status code is OK).
- the ID-WSF EPR was generated from the expected SvcMD elements (the unique address *http;//6-calendars.testing.com* shows this).

3.13.22. Query Calendar Service w/Action

Query for a Calendar Service including an `<Action>` element in the request.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
    <disco:Action>urn:x-test:cal:2008-03:GetMeeting</disco:Action>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- the action specified on the request is **not** explicitly listed in the registered SvcMD.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
       xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
       notOnOrAfter="2006-04-06T17:41:01Z"
       wsu:Id="EPRIDq9XReiwZpP_tftRcutoP">
    <wsa:Address>http://6-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
             usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
               ID="CREDst8YWUPHbqUdMKjnV_I7"
               IssueInstant="2006-04-06T15:41:01Z"
               Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- the returned ID-WSF EPR is essentially identical to the ID-WSF EPR returned in the previous query (only differing in timestamps and element IDs).
- the `<Action>` specified on the request was considered to be matched because **no** `<Action>` elements were specified in the registered SvcMD – which by definition means that the SvcMD matches all possible `<Action>` values.

3.13.23. Query Calendar Service w/resultsType=all

Query for the Calendar Service specifying the "...:TLS:SAMLV2" SecurityMechID and resultsType=all

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="all">
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- resultsType setting of "all" indicates that the requestor wants all possible results, not just a limited match. This is typically done when the client wants to choose which of the results to use.
- This query was constructed to be resolvable only by the data within the 1st `<EndpointContext>` element (the SecurithMechID *...:TLS:SAMLV2*) in the 2nd `<ServiceContext>` element (the ServiceType "*urn:x-test:cal:2008-03*").

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:05Z" wsu:Id="EPRIDgNpGPkjwRsZVIC63VOMt">
    <wsa:Address>https://4-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDYA0WkksXWLutGxWWsF2m"
              IssueInstant="2006-04-06T15:41:06Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:06Z" wsu:Id="EPRIDeB1MvrPuq-TphWEqQ7G0">
    <wsa:Address>https://5-calendars.testing.backup.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CRED8l6B4EvhFszmGOlAb5F7"
              IssueInstant="2006-04-06T15:41:06Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
 </disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- as expected 2 ID-WSF EPRs were returned because the two endpoints which matched the search criteria cannot be placed into the same EPR.

3.13.24. Query for all Calendar Service EPRs

A query of all of the data available for the calendar service

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
 <disco:RequestedService resultsType="all">
  <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
  <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
  <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
  <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
  <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
  <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
  <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
  <disco:SecurityMechID>urn:liberty:security:2005-02:null:Bearer</disco:SecurityMechID>
 </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- All of the service types and all of the SecurityMechIDs in the Calendar Service SvcMD are specified.
- The `resultsType` attribute is set to "*all*" indicating that the Discovery Service should return all possible results.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
 <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
 <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
     xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
     notOnOrAfter="2006-04-06T17:41:09Z" wsu:Id="EPRIDC08vEmUOfarryqg3zo8j">
  <wsa:Address>https://1-calendars.testing.com</wsa:Address>
  <wsa:Metadata>
   <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
   <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.1"/>
   <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
   <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
   <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
   <disco:SecurityContext>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
    <sec:Token xmlns:sec="urn:liberty:security:2006-08"
       usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
     <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
        ID="CRED1QVGIZghandZqdE90QRa"
        IssueInstant="2006-04-06T15:41:09Z" Version="2.0">
      ... assertion data was here ...
     </sa:Assertion>
    </sec:Token>
   </disco:SecurityContext>
   <disco:SecurityContext>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
   </disco:SecurityContext>
```

```
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:09Z" wsu:Id="EPRIDqEcv9d0T43OTQLdhB116">
    <wsa:Address>https://2-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      ... Metatdata was here ....
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:09Z" wsu:Id="EPRIDRPIps8-wMwgf4A_7DsHS">
    <wsa:Address>http://3-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      ... Metatdata was here ....
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:09Z" wsu:Id="EPRIDkLl5ahJdrutDV6gMPLyr">
    <wsa:Address>https://4-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      ... Metatdata was here ....
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:09Z" wsu:Id="EPRID0TmCLBCVoYZV7_R8jgky">
    <wsa:Address>https://5-calendars.testing.backup.com</wsa:Address>
    <wsa:Metadata>
      ... Metatdata was here ....
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:09Z" wsu:Id="EPRID9nsZ7UfG0G5UMMymJLRp">
    <wsa:Address>http://6-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      ... Metatdata was here ....
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- Much of the data in this response was elided in order to not waste alot of paper. The first EPR in the response is shown fairly completely.
- There are 6 EPRs (the minimum way to represent all of the data in the SvcMD that matched the requesed parameters).
- The first 3 EPRs have two service types (for the 2006-01 and the 2006-09 versions of the calendar service) as a single ID-WSF EPR may have multiple service types if they are all releated to the same logical service.

3.13.25. Query for one Calendar Service EPRs

A query for the first EPR out of all those within the Calendar service (to show the impact of the `resultsType` setting of *only-one*.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="only-one">
```

```
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
    <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:null:Bearer</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- All of the service types and all of the SecurityMechIDs in the Calendar Service SvcMD are specified.
- The `resultsType` attribute is set to "*only-one*" indicating that the Discovery Service should only return the first matching ID-WSF EPR.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:14Z" wsu:Id="EPRIDVh71zxFCQu4yWBOKLPzH">
    <wsa:Address>https://1-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.1"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
      <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDYjkZsJiWfQwKf5gaJoze"
              IssueInstant="2006-04-06T15:41:14Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- Only one ID-WSF EPR was returned (because of the query parameters) (as compared to the 6 returned in the previous example with the same query other than the `resultsType` attribute).
- The one ID-WSF EPR that is returned is the *first* EPR that would have otherwise been returned.

3.13.26. Query specific version of Calendar Service

A query for the 2006-09 version of the Calendar Service.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- This query was constructed to be resolvable only by the data within the 1st `<ServiceContext>` element (it's the only one with the ServiceType "*urn:x-test:cal:2006-09*") and within there, both the 1st and the 2nd `<EndpointContext>` element (they are the only ones with the SecurithMechID *...:TLS:Bearer*).

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:19Z" wsu:Id="EPRIDVsvR1CWhheodUvbDFelh">
    <wsa:Address>https://1-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.1"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDFy-6w5S8iOVearNxJur_"
              IssueInstant="2006-04-06T15:41:19Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:19Z" wsu:Id="EPRIDDKWLXiXVXpSJJi3oY3ge">
    <wsa:Address>https://2-calendars.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDW_y92CoMn7x57YOwZbnB"
              IssueInstant="2006-04-06T15:41:20Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- Two ID-WSF EPRs were returned by the Discovery Service because the matching data had two different endpoints which must be represented in separate EPRs.
- Because there was no `resultsType` specified on the request, the DS could have returned just the first ID-WSF EPR if they chose to as that ID-WSF EPR meets the basic requirements of the request. In this particular instance the Discovery Service acted as if "all" had been specified, but that should not be depended upon.

  If you need a particular `resultsType` behavior, you need to specify it on the request.
- The Discovery Service could have used the same assertion for the two ID-WSF EPRs if appropriate. In such a case, the one of the ID-WSF EPRs would have had a `<sec:Token>` element with a `ref` attribute containing a pointer to the `<sec:Token>` in the other ID-WSF EPR.

3.13.27. SvcMDReplace Calendar Service

Replace the SvcMD for the Calendar service with a much simpler SvcMD.

```
<disco:SvcMDReplace xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMD svcMDID="SVCMDIDMfPMb1wcRSiwnu8D3BGO">
    <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
    <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
    <disco:ServiceContext>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:EndpointContext>
        <disco:Address>https://calendar.testing.com</disco:Address>
        <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
      </disco:EndpointContext>
    </disco:ServiceContext>
  </disco:SvcMD>
</disco:SvcMDReplace>
```

Things to note about this query:

- the SvcMDID is the SvcMDID for the complex Calendar Service SvcMD that we have been querying over the past few requests.
- the new SvcMD is much simpler.

Server Response:

```
<disco:SvcMDReplaceResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDReplaceResponse>
```

Things to note about this response:

- the replacement was successful (status code is OK).

3.13.28. Query old Calendar Service

Query the Calendar Service data that was replaced.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService>
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- this is the same query we ran earlier that obtained results.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="Failed">
    <lu:Status code="NoResults"/>
  </lu:Status>
</disco:QueryResponse>
```

Things to note about this response:

- the query failed (status code is Failed)
- the sub-status is NoResults - indicating that no matching data was found
- The new (replacement) SvcMD for the Calendar Service has taken effect for the principal without the need for it to be associated with the principal (as it is already associated).

3.13.29. Query for all Calendar Service EPRs

A query of all of the data available for the calendar service (same query we ran a few requests ago).

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="all">
    <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
    <disco:ServiceType>urn:x-test:cal:2006-01</disco:ServiceType>
    <disco:ServiceType>urn:x-test:cal:2006-09</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:null</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2006-08:null:SAMLV2</disco:SecurityMechID>
    <disco:SecurityMechID>urn:liberty:security:2005-02:null:Bearer</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- All of the service types and all of the SecurityMechIDs in the Calendar Service SvcMD are specified.
- The `resultsType` attribute is set to "*all*" indicating that the Discovery Service should return all possible results.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
       xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
       notOnOrAfter="2006-04-06T17:41:32Z" wsu:Id="EPRID23j-JGWAXusqiV80ARzC">
    <wsa:Address>https://calendar.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test Calendar Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:cal:2008-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
             usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
               ID="CREDxHDqbb4F1TV7jSYx1mxq"
               IssueInstant="2006-04-06T15:41:32Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- Only the data from the replacement SvcMD is represented in the one ID-WSF EPR in the results.

3.13.30. SvcMDAssociationAdd the ATM Service SvcMD

Associate the ATM Service SvcMD with the current principal.

```
<disco:SvcMDAssociationAdd xmlns:disco="urn:liberty:disco:2006-08">
  <disco:SvcMDID>SVCMDIDrpG8SJpeUdSmla_ZSFUN</disco:SvcMDID>
</disco:SvcMDAssociationAdd>
```

Things to note about this query:

- the SvcMDID specified is the SvcMDID assigned to the test ATM service registered above (the third SvcMD in the multi-SvcMD registration that was done earlier).

Server Response:

```
<disco:SvcMDAssociationAddResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
</disco:SvcMDAssociationAddResponse>
```

Things to note about this response:

- the association was successful (status code is OK).
- the ATM Service SvcMD is now associated with the principal and available to subsequent Discovery Service queries.

3.13.31. Query ATM Service w/resultsType=best

Query for the ATM Service with the `resultsType` setting of "*best*."

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="best">
    <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- No `<disco:Action>`s are specfied which implies the caller wants to have access to all operations at the provider.

  This is important for the ATM Service because in the SvcMD, each endpoint was registered with a subset of actions (no one endpoint has them all, so the rewults will take several EPRs).

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:43Z" wsu:Id="EPRIDfmT9HOSbmMs3jZ1_qSuY">
    <wsa:Address>https://test2.atm.CA.US.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test ATM Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDS8x8pCKTOzaQMI14fkel"
              IssueInstant="2006-04-06T15:41:44Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
      <disco:Options>
        <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt3</disco:Option>
      </disco:Options>
      <disco:Action>urn:x-test:atm:2007-11:GetBalance</disco:Action>
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:44Z" wsu:Id="EPRIDdHYSEKB8_w5bPicTPe8o">
    <wsa:Address>https://readers.atm.CA.US.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test ATM Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CRED4nQ1FHP3vzuoqRhTMZRf"
              IssueInstant="2006-04-06T15:41:44Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
      <disco:Options>
        <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt3</disco:Option>
      </disco:Options>
      <disco:Action>urn:x-test:atm:2007-11:GetBalance</disco:Action>
      <disco:Action>urn:x-test:atm:2007-11:ListAccounts</disco:Action>
    </wsa:Metadata>
  </wsa:EndpointReference>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:44Z" wsu:Id="EPRIDggifVjR-zSAxkokPyfCo">
    <wsa:Address>https://writers.atm.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test ATM Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
              ID="CREDZNTlblH6VxPNrpwawtF6"
              IssueInstant="2006-04-06T15:41:44Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
      <disco:Options>
        <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt3</disco:Option>
      </disco:Options>
      <disco:Action>urn:x-test:atm:2007-11:Withdraw</disco:Action>
      <disco:Action>urn:x-test:atm:2007-11:Transfer</disco:Action>
    </wsa:Metadata>
  </wsa:EndpointReference>
  </disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- It took three ID-WSF EPRs to represent the set of actions at the ATM Service.
- One might think that because the "...GetBalance" action is on both the 1st and 2nd ID-WSF EPR and it is the only action on the 1st ID-WSF EPR, the results could have excluded that ID-WSF EPR and the client would still get to all of the resources at the ATM Service.

  However, the WSP placed the 1st "...GetBalance" action in the first `<EndpointContext>` and therefore gives it a higher priority in the results.
- The Discovery service could have left off the "...GetBalance" action on the 2nd ID-WSF EPR but that wouldn't have been much of a savings and so it was included. Clients should NOT depend upon this type of behavior.
- Even though Options were not specfied on the request, they are specified in the SvcMD and so are included in the ID-WSF EPRs generated from that SvcMD.

3.13.32. Query ATM Service w/Withdraw Action

Query for the ATM Service where "...Withdraw" action is available.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="all">
    <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:Action>urn:x-test:atm:2007-11:Withdraw</disco:Action>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- The `<disco:Action>` element is specified with the "*urn:x-test:atm:2007-11:Withdraw*") action value. So the client only intends to use this operation at the ATM Service.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
      notOnOrAfter="2006-04-06T17:41:52Z" wsu:Id="EPRIDkueZCk5N4IpSmX-0BD-9">
    <wsa:Address>https://writers.atm.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test ATM Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
             usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
               ID="CREDYfLT1S7tcZ8LkuU-z1rs"
               IssueInstant="2006-04-06T15:41:52Z" Version="2.0">
            ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
      <disco:Options>
        <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt3</disco:Option>
      </disco:Options>
      <disco:Action>urn:x-test:atm:2007-11:Withdraw</disco:Action>
      <disco:Action>urn:x-test:atm:2007-11:Transfer</disco:Action>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- Only one ID-WSF EPR was returned which contained the endpoint where the "...Withdraw" action is available.
- The Discovery service could have left off the "...Transfer" action but that wouldn't have been much of a savings and so it was included. Clients should NOT depend upon this type of behavior.
- Even though Options were not specfied on the request, they are specified in the SvcMD and so are included in the ID-WSF EPRs generated from that SvcMD.

3.13.33. Query ATM Service w/Option

Query for the ATM service specifying an option

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="only-one">
    <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
    <disco:Options>
      <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
    </disco:Options>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- The `resultsType` attribute is set to "*only-one*" indicating that the Discovery Service should only return the first matching ID-WSF EPR.
- the `<Option>` element is included with one of the values present in the SvcMD for the ATM Service.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"/>
  <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
       xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
       notOnOrAfter="2006-04-06T17:42:19Z" wsu:Id="EPRIDzKyaJ_h5Qb2jLjLjq59E">
    <wsa:Address>https://test2.atm.CA.US.testing.com</wsa:Address>
    <wsa:Metadata>
      <disco:Abstract>TestDisco Test ATM Service</disco:Abstract>
      <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
      <disco:ProviderID>https://s-wsp.liberty-iop.org:8743/sp.xml</disco:ProviderID>
      <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        <sec:Token xmlns:sec="urn:liberty:security:2006-08"
            usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          <sa:Assertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
            ID="CRED0FAqvGgoeySpLTZHbJa7"
            IssueInstant="2006-04-06T15:42:19Z" Version="2.0">
           ... assertion data was here ...
          </sa:Assertion>
        </sec:Token>
      </disco:SecurityContext>
      <disco:Options>
        <disco:Option>urn:x-test:atm:options:testopt1</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
        <disco:Option>urn:x-test:atm:options:testopt3</disco:Option>
      </disco:Options>
      <disco:Action>urn:x-test:atm:2007-11:GetBalance</disco:Action>
    </wsa:Metadata>
  </wsa:EndpointReference>
</disco:QueryResponse>
```

Things to note about this response:

- the query was successful (status code is OK).
- The one option specfied in the request matched one of the options specified in the SvcMD, so that is considerd a match. The caller does not have to specify all of the options in the SvcMD (but the SvcMD does have to have all of the options listed on the request).

- Even though only one option was specfied on the request, all of the options listed in the SvcMD are included in the response.

3.13.34. Query ATM Service w/unknown Option

Query for the ATM service with an option that doesn't exist in the SvcMD.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="all">
    <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
    <disco:Options>
      <disco:Option>urn:x-test:atm:options:testopt8</disco:Option>
    </disco:Options>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- The option specified is **not** in the ATM Service SvcMD.

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="Failed">
    <lu:Status code="NoResults"/>
  </lu:Status>
</disco:QueryResponse>
```

Things to note about this response:

- the query failed (status code is Failed).

- The sub-status was "NoResults" indicating no data matched the requested parameters

3.13.35. Query ATM Service w/good and bad Option

Query for the ATM service with an option that does exist and an option that doesn't exist in the SvcMD.

```
<disco:Query xmlns:disco="urn:liberty:disco:2006-08" id="discReq">
  <disco:RequestedService resultsType="all">
    <disco:ServiceType>urn:x-test:atm:2003-03</disco:ServiceType>
    <disco:Options>
      <disco:Option>urn:x-test:atm:options:testopt2</disco:Option>
      <disco:Option>urn:x-test:atm:options:testopt8</disco:Option>
    </disco:Options>
    <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
  </disco:RequestedService>
</disco:Query>
```

Things to note about this query:

- both specified options have to be available for this request to be considered matched (if, on the other hand, the request had the two `<disco:Option>` elements in separate `<disco:Options>` containers, a SvcMD could match either one).

Server Response:

```
<disco:QueryResponse xmlns:disco="urn:liberty:disco:2006-08">
  <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="Failed">
    <lu:Status code="NoResults"/>
  </lu:Status>
</disco:QueryResponse>
```

Things to note about this response:

- the query failed (status code is Failed).
- The sub-status was "NoResults" indicating no data matched the requested parameters

4. Discovery Service ID-WSF EPR conveyed via a Security Token

In both single sign-on and web services environments, many recipients of a security token find the need to subsequently invoke the identified principal's Discovery Service in order to discover and invoke identity services on behalf of said principal. For example, a SAML SP upon receiving an SSO assertion may want to discover and invoke the principals Profile Service and would need the Discovery Service ID-WSF EPR in order to do so.

In the SSO environment, this concept is often referred to as the "Discovery Service Bootstrap" in that the SP is using the data in the SSO assertion to bootstrap into the ID-WSF environment.

The need for this Discovery Service ID-WSF EPR is not restricted to SSO environments as any WSP that is invoked by a WSC may in turn need to act as a WSC and invoke other WSPs in order to fulfill the requested operation. For example, a Profile Service WSP may need to invoke the Interaction Service in order to request consent from the user before releasing data to a WSC.

This section describes the recommended interoperable method for an Identity Provider and/or Discovery Service can embed an ID-WSF EPR for the Discovery Service within security and/or Identity tokens that they issue. Unfortunately, because of the variance in structure and formats of various tokens, the model used tends to be specific to the format of the security token. The remainder of this section documents how this is accomplished within some specific token formats.

4.1. EPR Generation Rules

The Discovery Service Bootstrap ID-WSF EPR which is placed into any security token must be generated according to the following rules:

- The `<wsa:EndpointReference>` that MAY contain `<SecurityContext>` element(s) in turn containing `<sec:Token>` elements containing embedded security tokens, which are necessary to access the Discovery Service instance(s).
- The `<sec:Token>` element MAY instead include a reference to an external security token using a `<wsse:SecurityTokenReference>` containing a non-relative URI reference to a security token.
- The `<sec:Token>` element's `ref` attribute MAY instead refer to local security token available elsewhere in the same security token (such as another ID-WSF EPR within the security token). These references SHOULD only refer to elements within the security token carrying the ID-WSF EPR so that the reference will remain valid if the security token is separated from any message carrying the token.

  It is even possible (and in some cases typical) for the reference to be to the enveloping security token itself (the security token that contains this ID-WSF EPR) In such cases, the enveloping security token SHOULD carry the necessary information to support its consumption at the Discovery Service (as well as the information necessary for consumption at its primary relying party (the SP/WSP)).

  For example, with a SAML Assertion, this includes:

  - A second `<Audience>` element with the Discovery Service's ProviderID.
  - A subject confirmation method that the relying party can meet. This will frequently be `urn:oasis:names:tc:SAML:2.0:cm:bearer` in which case the same confirmation can be used by both parties. However, the assertion could contain multiple confirmation methods one for the initial party to use when invoking the relying party and one for the relying party to use when invoking the DS.

  This will allow the Discovery Service to validate the assertion using the normal assertion processing rules without having to manage some form of exception for self issued assertions.

4.2. SAML 2.0 Security Tokens

In a SAML 2.0 Assertion, the Discovery Service ID-WSF EPR SHOULD be conveyed as an XML element within the `<saml2:AttributeStatement>` element in a `<saml2:Assertion>`.

The `<saml2:AttributeStatement>` SHOULD be constructed according to the following rules:

- The `Name` attribute of the `<saml2:Attribute>` element MUST be:

  urn:liberty:disco:2006-08:DiscoveryEPR

- The `NameFormat` attribute of the `<saml2:Attribute>` element MUST be:

  urn:oasis:names:tc:SAML:2.0:attrname-format:uri

- One or more `<saml2:AttributeValue>` elements MUST be included which each containing a single `<wsa:EndpointReference>` element identifying a Discovery Service instance(s). These Discovery Service instances SHOULD offer identity services for the Principal identified in the Subject element inside the `<saml2:Assertion>`.

An example `<saml2:AttributeStatement>` that might be found in a SAMLv2 `<saml2:Assertion>` follows. The example includes a `<sec:Token>` element which has a reference to the surrounding assertion.

```

<AttributeStatement xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Attribute Name="urn:liberty:disco:2006-08:DiscoveryEPR"
           NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <AttributeValue>
      <wsa:EndpointReference>
        <wsa:Address>https://example.com/disco/</wsa:Address>

        <wsa:Metadata>
          <Abstract>
              The Principal’s Discovery Service  Resource
          </Abstract>

          <ServiceType>urn:liberty:disco:2006-08</ServiceType>

          <ProviderID>http://example.com/</ProviderID>

          <SecurityContext>
            <SecurityMechID>urn:liberty:security:2005-02:TLS:bearer</SecurityMechID>
            <sec:Token ref="..." usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
          </SecurityContext>
        </wsa:Metadata>
      </wsa:EndpointReference>
    </AttributeValue>
  </Attribute>
</AttributeStatement>

```

Example 21. `<AttributeStatement>` that might be found in a SAMLv2 AuthnResponse

In all cases, this `<AttributeStatement>` MUST carry an ID-WSF EPR for the Liberty Discovery Service. Any other ID-WSF EPRs are to be discovered by contacting the Discovery Service.

4.3. SAML 1.x (Liberty ID-FF) Security Tokens

In a SAML 1.x Assertion, the Discovery Service ID-WSF EPR SHOULD be conveyed as an XML element within the `<saml:AttributeStatement>` element in a `<saml:Assertion>`.

The `<saml:AttributeStatement>` SHOULD be constructed according to the following rules:

- For the `<saml:Attribute>` element:
    - The `AttributeName` attribute MUST be "*DiscoveryEPR*."
    - The `AttributeNamespace` attribute MUST be "*urn:liberty:disco:2006-08*."
- The `<Subject>` element of the `<saml:AttributeStatement>` element MUST carry the identity of the principal whose Discovery Service is referenced by this EPR and SHOULD be the same identity in the subject of the other statements in the `<saml:Assertion>`.
- One or more `<saml:AttributeValue>` elements MUST be included which each containing a single `<wsa:EndpointReference>` element identifying a Discovery Service instance(s). These Discovery Service instances SHOULD offer identity services for the Principal identified in the Subject element inside this `<saml:AttributeStatment>`.

An example `<saml:AttributeStatement>` that might be found in a SAML 1.1 `<saml:Assertion>` follows. The example includes a `<sec:Token>` element which has a reference to the surrounding assertion.

```

<AttributeStatement xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
 <Subject>
   <NameIdentifier Format="urn:liberty:iff:nameid:federated">
     d0CQF8elJTDLmzE0
   </NameIdentifier>
 </Subject>
 <Attribute AttributeName="DiscoveryEPR"
         AttributeNamespace="urn:liberty:disco:2006-08">
   <AttributeValue>
     <wsa:EndpointReference>
       <wsa:Address>https://example.com/disco/</wsa:Address>

       <wsa:Metadata>
         <Abstract>
            The Principal’s Discovery Service  Resource
         </Abstract>

         <ServiceType>urn:liberty:disco:2006-08</ServiceType>

         <ProviderID>http://example.com/</ProviderID>

         <SecurityContext>
           <SecurityMechID>urn:liberty:security:2005-02:TLS:bearer</SecurityMechID>
           <sec:Token ref="..." usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
         </SecurityContext>
       </wsa:Metadata>
     </wsa:EndpointReference>
   </AttributeValue>
 </Attribute>
</AttributeStatement>

```

Example 22. `<AttributeStatement>` that might be found in a SAML 1.1 AuthnResponse

In all cases, this `<AttributeStatement>` MUST only carry an ID-WSF EPR for the Liberty Discovery Service. Any other ID-WSF EPRs are to be discovered by contacting the Discovery Service.

5. ID-WSF 1.x Resource Offering conveyed in an EPR

In order to support the discovery and subsequent invocation of ID-WSF 1.0 and 1.1 services it may be necessary for the Discovery Service to carry the ID-WSF 1.x Resource Offering information within the ID-WSF EPR.

The process involves taking the fields that would normally be present in the Resource Offering and placing them into the appropriate fields within the EPR according to the following rules:

- The `<ResourceID>` element and/or the `<EncryptedResourceID>` element are placed into the `<Metadata>` element as-is.
- The `<ServiceType>` element in the `<ServiceInstance>` element is placed into the `<Metadata>` element.
- The `<ProviderID>` element in the `<ServiceInstance>` element is placed into the `<Metadata>` element.
- The `<SecurityMechID>` element in `ServiceInstance/Description` is placed into the `<SecurityContext>` element (and will be combined with other SecurityMechIDs based upon whether or not they share the same endpoint *and* credential (or do not use a credential)).
- The data from the `<Endpoint>` element in `ServiceInstance/Description` is placed into the `<Address>` element. Note that if there are multiple distinct `<Endpoint>`s they must be placed into different ID-WSF EPRs rather than being able to be placed into a single EPR like they were in an RO.
- Options are placed into the `<Metadata>` element.
- Abstract is placed into the `<Metadata>` element.
- Credentials, which in the days of the Resource Offering were carried elsewhere in the message and referenced from the `ServiceInstance/Description` element are now carried directly within the EPR in a `<sec:Token>` element in the `<SecurityContext>` element.

In addition, the ID-WSF EPR MUST also include at least one `<sbf:Framework>` element with the appropriate value (1.0 or 1.1) in the `version` attribute for the ID-WSF version being used.

As an example, let's start with an example ID-WSF 1.x Resource Offering:

```

<ResourceOffering>
  <ResourceID> 123 </ResourceID>
  <ServiceInstance>
    <ServiceType>urn:liberty:idsis-pp:2003-08</ServiceType>
    <ProviderID>http://pp.services.aol.com</ProviderID>
    <Description CredentialRef="1">
      <SecurityMechID>urn:liberty:security:2006-08:TLS:Bearer</SecurityMechID>
      <Endpoint>https://ep1.pp.service.aol.com</Endpoint>
    </Description>
    <Description>
      <SecurityMechID>urn:liberty:security:2006-08:Client-TLS:Null</SecurityMechID>
      <Endpoint>https://ep1.pp.service.aol.com</Endpoint>
    </Description>
  </ServiceInstance>
</ResourceOffering>
<Credentials>
  <saml1:Assertion AssertionID="1" ...>
    Assertion data goes here
  </saml1:Assertion>
</Credentials>

```

Translating this using the above rules would result in the following ID-WSF EPR:

```
<wsa:EndpointReference>
  <wsa:Address>https://ep1.pp. services.aol.com</wsa:Address >
  <wsa:Metadata>
    <ds1:ResourceID>123</ds1:Resourc eID>
    <ds2:ProviderID>http://pp.services.aol.com</ds2: ProviderID>
    <ds2:ServiceType>urn:liberty:idsis-pp: 2003-08</ds2:ServiceType>
    <ds2:Framework Version="1.1" />
    <ds2:SecurityContext>
      <ds2:SecurityMechID>urn:liberty:security:2006- 08:TLS:Bearer</ds2:Security MechID>
      <sec:Token usage="urn:liberty:security:to kenusage:2006-08:SecurityTok en">
        <saml1:Assertion AssertionID="1" ... >
          ... assertion data goes here ...
        </saml1:Assertion>
      </sec:Token>
    </ds2:SecurityContext>
    <ds2:SecurityContext>
      <ds2:SecurityMechID>
        urn:liberty:security:2006-08:Client-TLS:Null
      </ds2:SecurityMechID>
    </ds2:SecurityContext>
  </wsa:Metadata>
</wsa:EndpointReference>
```

And subsequently, the invocation of the ID-WSF 1.x service would look to be something along the lines of (assuming that the WSC chose to use the "...:TLS:bearer" Security Mechanism):

```
<?xml version="1.0" encoding="utf-8" ?>
<S:Envelope....
  <S:Header>
    <sb:Correlation S:mustUnderstand="1"
       messageID=uuid:958312848-29348938-232342121
       timestamp="2003-06-06T18:29:18Z"  />
    <wsse:Security>
       <saml1:Assertion AssertionID="1" ... >
          ... assertion data goes here ...
       </saml1:Assertion>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <pp:Query>
       <pp:ResourceID>123</pp:ResourceID>
       <pp:QueryItem>
          ... query data goes here ...
       </pp:QueryItem>
    </pp:Query>
  </S:Body>
</S:Envelope>
```

6. Acknowledgments

Many people have made contributions to this specification as it has evolved over time. The original specification was written by John Beatty with subsequent versions "inked" by Jonathan Sergent and later, Jeff Hodges and now myself.

The changes made in this latest release of the specification are due in a large part to the work of Jeff Hodges, Robert Aarts, John Kemp, Gary Ellison and Greg Whitehead. Many others, including those that are listed as contributors on the cover page, have also played a part in this and earlier releases of the specification. Many thanks to all who participated (and apologies if I have forgotten to mention your name).

References

Normative

[LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertySecMech11] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.1, Liberty Alliance Project (18 April 2004). *http://www.projectliberty.org/specs/*

[LibertySecMech20SAML] Hirsch, Frederick, eds. "ID-WSF 2.0 SecMech SAML Profile," Version 2.0-errata-v1.0, Liberty Alliance Project (08 November, 2006). *http://www.projectliberty.org/specs*

[LibertyAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. " Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification ," Version 2.0-errata-v1.0, Liberty Alliance Project (28 November, 2006). *http://www.projectliberty.org/specs*

[LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0 errata v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata," Version 1.0, Liberty Alliance Project (13 April, 2007). *http://www.projectliberty.org/specs*

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

[RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also STD0066) (Status: STANDARD), The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3986.txt*

[SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf*

[SAMLMeta2] Cantor, Scott, Moreh, Jahan, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf*

[Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-1/*

[WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). *http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/*

[WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). *http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/*

[WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith, Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001). *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*

[wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf*

[wss-sms-draft] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (June 30, 2003). "Web Services Security: SOAP Message Security," Draft WSS-SOAPMessageSecurity-14-06-2003, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/download.php/2757/WSS-SOAPMessageSecurity-14-063003-merged.pdf*

[xmlenc-core] Eastlake, Donald, Reagle, Joseph, eds. (10 December 2002). "XML Encryption Syntax and Processing," W3C Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlenc-core/*

Informative

[LibertyPAOS] Aarts, Robert, Kemp, John, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyClientProfiles] Aarts, Robert, Kainulainen, Jukka, Kemp, John, eds. "Liberty ID-WSF Profiles for Liberty-Enabled User Agents and Devices," Version 2.0-errata-v1.0, Liberty Alliance Project (22 January, 2007). *http://www.projectliberty.org/specs*

A. Discovery Service Version 2.0 XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:disco:2006-08"
   xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
   xmlns:sb="urn:liberty:sb:2006-08"
   xmlns:sbf="urn:liberty:sb"
   xmlns:sec="urn:liberty:security:2006-08"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"

   xmlns="urn:liberty:disco:2006-08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>

  <xs:import namespace="urn:liberty:util:2006-08"
   schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>

  <xs:import namespace="urn:liberty:sb:2006-08"
   schemaLocation="liberty-idwsf-soap-binding-v2.0.xsd"/>

  <xs:import namespace="urn:liberty:sb"
   schemaLocation="liberty-idwsf-soap-binding.xsd"/>

  <xs:import namespace="http://www.w3.org/2005/08/addressing"
   schemaLocation="ws-addr-1.0.xsd"/>

  <xs:import namespace="urn:oasis:names:tc:SAML:2.0:metadata"
   schemaLocation="saml-schema-metadata-2.0.xsd"/>

  <xs:import namespace="urn:liberty:security:2006-08"
   schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>

  <xs:import
    namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    schemaLocation="wss-secext-1.0.xsd"/>

  <xs:import
    namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
"
    schemaLocation="wss-util-1.0.xsd"/>

  <!-- **** Discovery Service Data Elements & Types **** -->

  <!--  The data elements and types in this section are used to
       embellish WS-Addressing Endpoint References (EPRs).
       They are placed in the /wsa:EndpointReference/Metadata
       element. Specific usage and cardinalities are stipulated
       in the Discovery Service v2.0 Specification.  -->

  <!-- Abstract: natural-language description of service  -->

  <xs:element name="Abstract" type="xs:string"/>

  <!-- Provider ID  -->

  <xs:element name="ProviderID" type="xs:anyURI"/>

  <!-- Service Type -->

  <xs:element name="ServiceType" type="xs:anyURI"/>
```

```
<!-- Framework Description -->

<xs:element name="Framework" type="sbf:FrameworkType" />

<!-- EPR Expiration Timestamp  -->

<xs:attribute name="notOnOrAfter" type="xs:dateTime"/>

<!--  Security Context Container  -->

<xs:element name="SecurityContext">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="SecurityMechID"
              minOccurs="1"
              maxOccurs="unbounded"/>

      <xs:element ref="sec:Token"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Security Mechanism ID -->

<xs:element name="SecurityMechID" type="xs:anyURI"/>

<!-- Options -->

<xs:element name="Options" type="OptionsType"/>

<xs:element name="Option" type="xs:anyURI" />

<xs:complexType name="OptionsType">
  <xs:sequence>
    <xs:element ref="Option" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<!-- Address -->

<xs:element name="Address" type="xs:anyURI"/>

<!-- Action(s) - the interfaces available at this service -->

<xs:element name="Action" type="xs:anyURI"  />
 <!-- Keys Element - For use in ModifyResponse -->

 <xs:element name="Keys" type="KeysType"/>

 <xs:complexType name="KeysType">
   <xs:sequence>
    <xs:element ref="md:KeyDescriptor"
            minOccurs="1"
            maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>

<!-- Service Metadata (SvcMD) - metadata about service instance -->

<xs:element name="SvcMD" type="SvcMetadataType"/>
<xs:complexType name="SvcMetadataType">
  <xs:sequence>
    <xs:element ref="Abstract"                     />
    <xs:element ref="ProviderID"                    />
```

```
      <xs:element ref="ServiceContext"  maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="svcMDID" type="xs:string" use="optional" />
  </xs:complexType>

  <!-- ServiceContext - describes service type/option/endpoint context -->
  <xs:element name="ServiceContext" type="ServiceContextType"/>
  <xs:complexType name="ServiceContextType">
    <xs:sequence>
      <xs:element ref="ServiceType"    maxOccurs="unbounded" />
      <xs:element ref="Options"       minOccurs="0"
                              maxOccurs="unbounded" />
      <xs:element ref="EndpointContext" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- EndpointContext - describes endpoints used to access service -->
  <xs:element name="EndpointContext" type="EndpointContextType" />
  <xs:complexType name="EndpointContextType">
    <xs:sequence>
      <xs:element ref="Address"      maxOccurs="unbounded" />
      <xs:element ref="sbf:Framework"  maxOccurs="unbounded" />
      <xs:element ref="SecurityMechID" maxOccurs="unbounded" />
      <xs:element ref="Action"      minOccurs="0"
                             maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- SvcMD ID element used to refer to Service Metadata elements -->
  <xs:element name="SvcMDID" type="xs:string" />

  <!-- **** Discovery Service Protocol Messages Elements & Types **** -->

  <!-- Query Message Element & Type -->

  <xs:element name="Query" type="QueryType"/>

  <xs:complexType name="QueryType">
    <xs:sequence>
      <xs:element name="RequestedService"
              type="RequestedServiceType"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>

    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="RequestedServiceType">
    <xs:sequence>
      <xs:element ref="ServiceType" minOccurs="0" maxOccurs="unbounded" />

      <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded" />

      <xs:element ref="Options" minOccurs="0" maxOccurs="unbounded"/>

      <xs:element ref="SecurityMechID" minOccurs="0" maxOccurs="unbounded"/>

      <xs:element ref="Framework" minOccurs="0" maxOccurs="unbounded"/>

      <xs:element ref="Action" minOccurs="0" maxOccurs="unbounded"/>

      <xs:any namespace="##other"
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>

    </xs:sequence>

    <xs:attribute name="reqID" type="xs:string" use="optional" />
    <xs:attribute name="resultsType" type="xs:string" use="optional" />

  </xs:complexType>

  <!-- QueryResponse Message Element & Type -->

  <xs:element name="QueryResponse" type="QueryResponseType"/>

  <xs:complexType name="QueryResponseType">
    <xs:sequence>
      <xs:element ref="lu:Status"/>

      <xs:element ref="wsa:EndpointReference"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>


  <!--                                -->
  <!-- DS Interfaces for SvcMD Associations       -->
  <!--                                -->
  <!-- These interfaces support the adding, deleting,-->
  <!-- querying SvcMD Associations for a principal.  -->
  <!--                                -->

  <!-- SvcMDAssociationAdd operation -->

  <xs:element name="SvcMDAssociationAdd" type="SvcMDAssociationAddType"/>

  <xs:complexType name="SvcMDAssociationAddType">
    <xs:sequence>
      <xs:element ref="SvcMDID" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <!-- Response for SvcMDAssociationAdd operation -->

  <xs:element name="SvcMDAssociationAddRespon se"
          type="SvcMDAssociationAddResponseType"/>

  <xs:complexType name="SvcMDAssociationAddResponseType">
    <xs:sequence>
      <xs:element ref="lu:Status" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <!-- SvcMDAssociationDelete operation -->

  <xs:element name="SvcMDAssociationDelete" type="SvcMDAssociationDeleteType "/>

  <xs:complexType name="SvcMDAssociationDeleteType">
    <xs:sequence>
      <xs:element ref="SvcMDID" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <!-- Response for SvcMDAssociationDelete operation -->

  <xs:element name="SvcMDAssociationDeleteRe sponse"
          type="SvcMDAssociationDeleteResponse Type"/>

```

```
<xs:complexType name="SvcMDAssociationDeleteResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
<!-- SvcMDAssociationQuery operation -->

<xs:element name="SvcMDAssociationQuery" type="SvcMDAssociationQueryType"/>

<xs:complexType name="SvcMDAssociationQueryType">
  <xs:sequence>
    <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
<!-- Response for SvcMDAssociationQuery operation -->

<xs:element name="SvcMDAssociationQueryResponse"
        type="SvcMDAssociationQueryResponseType"/>

<xs:complexType name="SvcMDAssociationQueryResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
    <xs:element ref="SvcMDID" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!--                                    -->
<!-- DS Interfaces for Service Metadata Management -->
<!--                                    -->
<!-- These interfaces document a create, replace,  -->
<!-- delete, and query interface for the service   -->
<!-- metadata which is later associated with a    -->
<!-- principal.                            -->
<!--                                    -->

<!-- Register operation for Service Metadata -->

<xs:element name="SvcMDRegister" type="SvcMDRegisterType"/>

<xs:complexType name="SvcMDRegisterType">
  <xs:sequence>
    <xs:element ref="SvcMD" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Response for SvcMDRegister operation -->

<xs:element name="SvcMDRegisterResponse"
        type="SvcMDRegisterResponseType"/>

<xs:complexType name="SvcMDRegisterResponseType">
  <xs:sequence>

    <xs:element ref="lu:Status" />
    <xs:element ref="SvcMDID"  minOccurs="0" maxOccurs="unbounded" />
    <xs:element ref="Keys"    minOccurs="0" maxOccurs="unbounded" />

  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Delete operation on Service Metadata -->

```

```
<xs:element name="SvcMDDelete" type="SvcMDDeleteType"/>

<xs:complexType name="SvcMDDeleteType">
  <xs:sequence>
    <xs:element ref="SvcMDID" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Response for delete operation on Service Metadata -->

<xs:element name="SvcMDDeleteResponse" type="SvcMDDeleteResponseType"/>

<xs:complexType name="SvcMDDeleteResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Query operation on Service Metadata -->

<xs:element name="SvcMDQuery" type="SvcMDQueryType"/>

<xs:complexType name="SvcMDQueryType">
  <xs:sequence>
    <xs:element ref="SvcMDID"
            minOccurs="0"
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Response for Query operation on Service Metadata -->

<xs:element name="SvcMDQueryResponse" type="SvcMDQueryResponseType"/>

<xs:complexType name="SvcMDQueryResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
    <xs:element ref="SvcMD" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Replace operation on Service Metadata -->

<xs:element name="SvcMDReplace" type="SvcMDReplaceType"/>

<xs:complexType name="SvcMDReplaceType">
  <xs:sequence>
    <xs:element ref="SvcMD" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- Response for SvcMDReplace operation -->

<xs:element name="SvcMDReplaceResponse" type="SvcMDReplaceResponseType"/>

<xs:complexType name="SvcMDReplaceResponseType">
  <xs:sequence>
    <xs:element ref="lu:Status" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

```
</xs:schema>
```

B. Discovery Service WSDL

```
<?xml version="1.0"?>
<definitions name="disco-svc"
  targetNamespace="urn:liberty:disco:2006-08"
  xmlns:tns="urn:liberty:disco:2006-08"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:sb="urn:liberty:sb:2006-08"
  xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
  xmlns:disco="urn:liberty:disco:2006-08"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
                    http://schemas.xmlsoap.org/wsdl/
    http://www.w3.org/2006/02/addressing/wsdl
    http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

  <types>
    <xsd:schema>
      <xsd:import namespace="urn:liberty:disco:2006-08"
              schemaLocation="liberty-idwsf-disco-svc-v2.0.xsd"/>
    </xsd:schema>
  </types>

  <message name="Query">
    <part name="body" element="disco:Query"/>
  </message>
  <message name="QueryResponse">
    <part name="body" element="disco:QueryResponse"/>
  </message>

  <message name="SvcMDAssociationAdd">
    <part name="body" element="disco:SvcMDAssociationAdd"/>
  </message>
  <message name="SvcMDAssociationAddResponse">
    <part name="body" element="disco:SvcMDAssociationAddResponse"/>
  </message>

  <message name="SvcMDAssociationQuery">
    <part name="body" element="disco:SvcMDAssociationQuery"/>
  </message>
  <message name="SvcMDAssociationQueryResponse">
    <part name="body" element="disco:SvcMDAssociationQueryResponse" />
  </message>

  <message name="SvcMDAssociationDelete">
    <part name="body" element="disco:SvcMDAssociationDelete" />
  </message>
  <message name="SvcMDAssociationDeleteResponse">
    <part name="body" element="disco:SvcMDAssociationDeleteResponse"/>
  </message>

  <message name="SvcMDRegister">
    <part name="body" element="disco:SvcMDRegister"/>
  </message>
  <message name="SvcMDRegisterResponse">
    <part name="body" element="disco:SvcMDRegisterResponse"/>
  </message>

  <message name="SvcMDQuery">
    <part name="body" element="disco:SvcMDQuery"/>
  </message>
  <message name="SvcMDQueryResponse">
    <part name="body" element="disco:SvcMDQueryResponse"/>
  </message>
```

```

<message name="SvcMDReplace">
  <part name="body" element="disco:SvcMDReplace"/>
</message>
<message name="SvcMDReplaceResponse">
  <part name="body" element="disco:SvcMDReplaceResponse"/>
</message>

<message name="SvcMDDelete">
  <part name="body" element="disco:SvcMDDelete"/>
</message>
<message name="SvcMDDeleteResponse">
  <part name="body" element="disco:SvcMDDeleteResponse"/>
</message>


<portType name="DiscoveryPort">

  <operation name="DiscoveryQuery">
    <input message="tns:Query"
       wsaw:Action="urn:liberty:disco:2006-08:Query" />
    <output message="tns:QueryResponse"
      wsaw:Action="urn:liberty:disco:2006-08:QueryResponse" />
  </operation>

  <operation name="MDAssociationAdd">
    <input message="tns:SvcMDAssociationAdd"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDAssociationAdd" />
    <output message="tns:SvcMDAssociationAddResponse"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDAssociationAddResponse" />
  </operation>

  <operation name="MDAssociationQuery">
    <input message="tns:SvcMDAssociationQuery"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDAssociationQuery" />
    <output message="tns:SvcMDAssociationQueryResponse"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDAssociationQueryResponse"/>
  </operation>

  <operation name="MDAssociationDelete">
    <input message="tns:SvcMDAssociationDelete"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDAssociationDelete" />
    <output message="tns:SvcMDAssociationDeleteResponse"
     wsaw:Action="urn:liberty:disco:2006-08:SvcMDAssociationDeleteResponse"/>
  </operation>

  <operation name="MetadataRegister">
    <input message="tns:SvcMDRegister"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDRegister" />
    <output message="tns:SvcMDRegisterResponse"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDRegisterResponse" />
  </operation>

  <operation name="MetadataQuery">
    <input message="tns:SvcMDQuery"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDQuery" />
    <output message="tns:SvcMDQueryResponse"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDQueryResponse" />
  </operation>

  <operation name="MetadataReplace">
    <input message="tns:SvcMDReplace"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDReplace" />
    <output message="tns:SvcMDReplaceResponse"
      wsaw:Action="urn:liberty:disco:2006-08:SvcMDReplaceResponse" />
  </operation>

```

```
    <operation name="MetadataDelete">
      <input message="tns:SvcMDDelete"
        wsaw:Action="urn:liberty:disco:2006-08:SvcMDDelete" />
      <output message="tns:SvcMDDeleteResponse"
        wsaw:Action="urn:liberty:disco:2006-08: SvcMDDeleteResponse" />
    </operation>


  </portType>

  <!--
  An example of a binding and service that can be used with this
  abstract service description is provided below.
  -->

  <binding name="DiscoveryBinding" type="tns:DiscoveryPort">

    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="DiscoveryQuery">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="MDAssociationAdd">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>


    <operation name="MDAssociationQuery">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="MDAssociationDelete">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="MetadataRegister">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="MetadataQuery">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="MetadataReplace">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="MetadataDelete">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>


  </binding>

  <service name="DiscoveryService">

    <port name="DiscoveryPort" binding="tns:DiscoveryBinding">

```

```
      <!-- Modify with the REAL SOAP endpoint -->

      <soap:address location="http://example.com/discovery"/>

    </port>

  </service>

</definitions>
```