
# Liberty ID-WSF Service Hosting and Proxying Service Specification

Version: 1.0

**Editors:**
Conor Cahill, Intel Corporation

**Contributors:**
Shelagh Callahan, Intel Corporation
Jane Dashevsky, Intel Corporation
Hubert Le Van Gong, Sun
Andrew Lindsay-Stewart, Vodafone
Tapio Kaukonen, Nokia
Sampo Kellomäki, Symlabs, Inc.
Paul Madsen, NTT
Paul Miller, Gemplus
Pierre Vannel, Gemplus
Sean Walker, Axalto
George Fletcher, AOL LLC
Hiroyoshi Takiguchi, NTT
Greg Whitehead, Hewlett-Packard

**Abstract:**

This specification describes the Service Hosting and Proxying Service and its interfaces.

**Filename:** liberty-idwsf-shps-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.



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

# Contents

# 1. Introduction

Smart clients are more and more capable of directly hosting identity services for the various service providers at which those clients interact. However, the realities of variable client connectivity and privacy concerns dictate that it may also be desirable that services also be hosted by network providers on behalf of such clients. A Service Hosting or Proxying Service (SHPS) provides such functionality to clients. This specification details the mechanisms by which clients can discover which services a SHPS is able to provide, request the SHPS provide particular services, and manage the availability of said services.

## 1.1. 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.

### 1.1.1. XML Namespaces

The following XML namespaces are referred to in this document:

- The prefix *shps:* represents the Service Hosting and Proxying 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 SHPS messages and fragments thereof, this is the default namespace when no prefix is shown:

  *urn:liberty:shps:2007-09*

- 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 *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. Overview

The Liberty ID-WSF Advanced Client Technologies Overview [LibertyACT] presents a complete overview of the provisioning process. The reader is strongly encouraged to read through that document (at least the provisioning section) prior to reading this document.

This document describes the Liberty ID-WSF Service Hosting/Proxying Service (SHPS). The SHPS provides a network visible endpoint for hosting and/or proxying service instances on behalf of a client service instance (CSI).

## 2.1. Client Service Components

The diagram below shows a typical set of interested parties in a client hosted service situation.

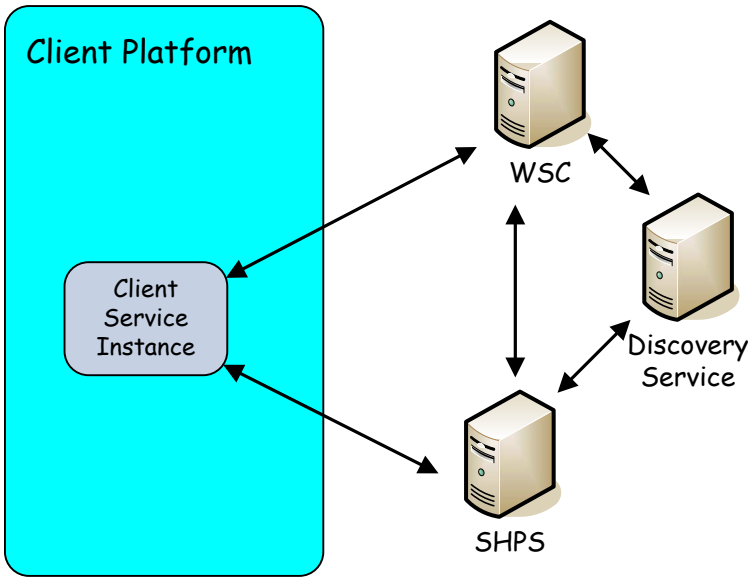


**Figure 1. CSI Actors**

The CSI is hosted on the client platform while the other parties, including the Web Services Consumer (WSC – the entity trying to invoke the service instance), Service Hosting/Proxying Service (SHPS) and the Discovery Service (DS), are typically applications hosted on network servers.

The Service Hosting/Proxying Service provides a means for the CSI to use a remote, network visible entity to expose/proxy its service. We'll talk more about that shortly.

The Discovery Service comes into play when discussing CSIs because service instances must be registered in the DS in order for them to be found by the WSC. Several of the operations surrounding the SHPS enablement will involve DS interactions.

This complexity is necessitated by a number of issues that must be considered when hosting a service on the client, including:

- Clients frequently have limited communications bandwidth (when compared to online services).
- Clients have more tenuous connectivity (being unavailable when the user goes through a tunnel or turns the device off for the night).

- Clients are frequently behind network firewalls, preventing incoming service invocation without modifications to the firewall rules.

- Multiple providers talking to the same service endpoint on the client for a particular user's service can use that service endpoint as a correlation handle for the user and potentially collude without user knowledge or control.

The user, and their client service instance, have the choice of the following hosting solutions:

- **Stand-alone** - the service is hosted exclusively on the client device and all service invokers must communicate directly with the device. This is the standard ID-WSF web services provider model where the service instance maintains a service metadata description at the Liberty ID-WSF Discovery Service (DS). Web service consumers (WSCs) discover and invoke the service instance using the Discovery Service.

  Such implementations must deal with or accept the connectivity and privacy issues outlined above. The PAOS protocol (see [LibertyPAOS] may be used with stand-alone service instances to resolve some of the connectivity issues.

  In most cases the stand-alone solution is used where collusion protection is not a concern and the service instance is on an always-connected device exposed directly on an external network.

- **Proxied** - the service instance is hosted exclusively on the client device, but uses the Liberty ID-WSF Service Hosting/Proxying service (SHPS) to proxy incoming calls. This solves two of the problems listed above: a) the privacy breaking cross-provider collusion concern is mitigated by the large number of client using the same SHPS service and b) the client doesn't have to have an externally exposed interface as it can poll the SHPS service for incoming request.

  In this case, the SHPS would be registered as the endpoint for the service instance for the user in the DS. WSCs would invoke the service instance at the SHPS and the SHPS would forward the request to the client, get the response back and forward the response to the WSC. The WSC would not be aware that the proxying is taking place.

  If the service proxy hosted at SHPS is invoked when the client instance is not available, the call fails as in this mode the SHPS is not configured to act in the name of the client.

- **Hosted** - a mirror of the service instance is hosted on the SHPS. Requests for service are handled directly by the hosted instance without additional interaction with the client instance. The client service instance keeps the hosted mirror service instance up-to-date as necessary.

  In this case, the SHPS would be registered as the endpoint for the service instance for the user in the DS. WSCs would invoke the service instance at the SHPS and SHPS would respond directly without involving the client.

- **Proxied+Hosted** - both hosting and proxying are implemented. Proxying is used when the client is available and when the client is not available SHPS is able to respond to the request using the data in it's mirrored service instance.

# 3. Service Hosting or Proxying Service Definition

A concrete definition of the SHPS interfaces.

An abstract WSDL definition for the SHPS is included in this document, see Section 5: Service Hosting/Proxying Service WSDL . This WSDL document defines all of the "WSDL operations" for the SHPS.

The complete schema for the SHPS is included in this document, see Section 4: Service Hosting/Proxying Service Schema .

## 3.1. Service URIs

**Table 1. SHPS URIs**

| Use | URI |
|---|---|
| Service Type | *urn:liberty:shps:2007-09* |
| Query wsa:Action | *urn:liberty:shps:2007-09:Query* |
| QueryResponse wsa:Action | *urn:liberty:shps:2007-09:QueryResponse* |
| QueryRegistered wsa:Action | *urn:liberty:shps:2007-09:QueryRegistered* |
| QueryRegisteredResponse wsa:Action | *urn:liberty:shps:2007-09:QueryRegisteredResponse* |
| Register wsa:Action | *urn:liberty:shps:2007-09:Register* |
| RegisterResponse wsa:Action | *urn:liberty:shps:2007-09:RegisterResponse* |
| Update wsa:Action | *urn:liberty:shps:2007-09:Update* |
| UpdateResponse wsa:Action | *urn:liberty:shps:2007-09:UpdateResponse* |
| Delete wsa:Action | *urn:liberty:shps:2007-09:Delete* |
| DeleteResponse wsa:Action | *urn:liberty:shps:2007-09:DeleteResponse* |
| Invoke wsa:Action | *urn:liberty:shps:2007-09:Invoke* |
| InvokeResponse wsa:Action | *urn:liberty:shps:2007-09:InvokeResponse* |
| GetStatus wsa:Action | *urn:liberty:shps:2007-09:GetStatus* |
| GetStatusResponse wsa:Action | *urn:liberty:shps:2007-09:GetStatusResponse* |
| SetStatus wsa:Action | *urn:liberty:shps:2007-09:SetStatus* |
| SetStatusResponse wsa:Action | *urn:liberty:shps:2007-09:SetStatusResponse* |
| Poll wsa:Action | *urn:liberty:shps:2007-09:Poll* |
| PollResponse wsa:Action | *urn:liberty:shps:2007-09:PollResponse* |
| ProxyInvoke wsa:Action | *urn:liberty:shps:2007-09:ProxyInvoke* |
| ProxyInvokeResponse wsa:Action | *urn:liberty:shps:2007-09:ProxyInvokeResponse* |

## 3.2. Data Definitions

### 3.2.1. Status Codes

All the response messages extended from `ResponseAbstractType` contain a `<lu:Status>` element (see Section 3.2.2.2) to indicate whether or not the processing of the request message has succeeded. The `<lu:Status>` element is included from the Liberty ID-WSF Utility Schema. A `<lu:Status>` element MAY contain other `<lu:Status>` elements providing more detailed information. A `<lu:Status>` element has a code attribute, which contains the return status as a string. The local definition of these codes is specified in this document. This specification defines the following status codes to be used as values for the code attribute:

- Failed
- Partial
- OK
- NeedServiceData
- NoResults
- NoSuchService

These strings are expected to appear in the "code" attribute of `<lu:Status>` elements used in SOAP-bound SHPS protocol messages [LibertySOAPBinding]. Specific uses for the status codes are defined in the processing rules for individual 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).

## 3.2.2. Request and Response Abstract Types

### 3.2.2.1. Complex Type RequestAbstractType

All request messages are of types that are derived from the abstract **`RequestAbstractType`** complex type. This type defines common attributes that are associated with all SHPS requests:

- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

The following schema fragment defines the **`RequestAbstractType`** complex type:

```
<!-- RequestAbstractType - common request message structure -->

<xs:complexType name="RequestAbstractType" abstract="true">
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

### 3.2.2.2. Complex Type ResponseAbstractType

All response messages are of types that are derived from the abstract **`ResponseAbstractType`** complex type. This type defines common attributes and elements that are associated with all SHPS responses:

- `<lu:Status>` **[Required]** - The `<lu:Status>` element is used to convey status codes and related information. The schema fragment is defined in the Liberty ID-WSF Utility schema. The local definition of status codes are described in Section 3.2.1.

- `anyAttribute` **[Optional]** - An attribute from a namespace other than that of this specification.

The following schema fragment defines the XML **ResponseAbstractType** complex type:

```
<!-- ResponseAbstractType - common message response structure -->

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

### 3.2.3. Service Mode

A service can be provided by a SHPS in one of three modes:

**Hosted** In this model, the SHPS has a full implementation of the service available and responds directly to any requests from WSCs. The SHPS must have the necessary local stores available for hosting the necessary information to expose the service in this manner.

This mode is identified by the URN *urn:liberty:shps:2007-09:svcmode:hosted*

**Proxied** In this model, the SHPS passes through any requests from WSCs to the CSI for processing and, of course, responds to the WSCs with the response it receives from the CSI.

This mode is identified by the URN *urn:liberty:shps:2007-09:svcmode:proxied*

**ProxyHosted** In this model, when the CSI is available to the SHPS, the SHPS passes through any requests from WSCs to the CSI for processing and, of course, responds to the WSCs with the response it receives from the CSI. When the CSI is not available, the SHPS is able to respond to the requester using the hosted service instance.

This mode is identified by the URN *urn:liberty:shps:2007-09:svcmode:proxyhosted*

A SHPS will advertise their ability to act (or not) in these modes in the `<ServiceInstance>` elements they create and return.

If a SHPS can or is willing to offer the service (as determined by its service type) in multiple modes, it MUST create and list different corresponding `<ServiceInstance>` elements.

### 3.2.4. Service Handle

A Service Handle is used to refer to an instance of a service exposed by the SHPS on behalf of an Advanced Client. This can be used to refer to any type of service in any Service Mode exposed by the SHPS.

The Service Handle is assigned by the SHPS during the service registration process (see Section 3.5) and is only usable by the entity which registered the service (i.e., the Service Handle is **not** transferable).

SHPS MUST ensure that there is a 1:1 match between a Service Handle and an exposed service instance. In other words, SHPS MUST NOT allow the creation of multiple service instances which cannot be disambiguated using the incoming invocation context (Target Identity, Service Type and Invocation Address (EPR Address)).

The identity of the entity which registers a service with the SHPS is taken from the invocation context of the registration call and is usually associated with a security token used in the context. It is possible that different CSIs within a given Advanced Client use the same identification and in such cases the handles for different hosted services at different CSIs could be associated with the same client identity. In other cases, each CSI could have a unique client identity and as such there would be a 1:1 relationship between the client identity and the hosted or proxied service instance.

The schema for the `<shps:ServiceHandle>` is shown below.

```
<!-- ServiceHandle - a reference to a hosted/proxied service instance -->

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

**Figure 2. `<shps:ServiceHandle>` — Schema Fragment**

An example `<shps:ServiceHandle>` is shown below.

```
<shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
```

**Example 1. Example `<shps:ServiceHandle>`**

### 3.2.5. InvocationContext

The `<shps:InvocationContext>` element is used in `<shps:ProxyInvoke>` calls to describe how the SHPS was invoked so that the Advanced Client can make the appropriate access control decisions for the data.

The invocation context data is built from the information in the SOAP headers provided to the SHPS when the proxied service instance at SHPS was invoked.

The `<shps:InvocationContext>` element has the following elements/attributes:

- `<shps:InvokingProvider>` : **[Required]** the ProviderID of the provider who invoked the service instance at the SHPS. This will usually be taken from the security token used to invoke the SHPS.
- `<shps:InvokingPrincipal>` : **[Optional]** the identity of the invoking party. This SHOULD NOT be specified when the invoker is the principal for which the service is registered.
- `<disco:SecurityMechID>` : **[Required]** the security mechanism ID that describes the security context used to invoke the service instance at the SHPS.

The schema for the `<shps:InvocationContext>` is shown below.

```
<!-- InvocationContext - how the proxied service instance was invoked -->

<xs:element name="InvocationContext" type="InvocationContextType" />

<xs:complexType name="InvocationContextType">
  <xs:sequence>
    <xs:element ref="InvokingProvider" />
    <xs:element ref="InvokingPrincipal" minOccurs="0" />
    <xs:element ref="disco:SecurityMechID" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="InvokingProvider"  type="xs:string" />
<xs:element name="InvokingPrincipal" type="saml2:NameIDType" />
```

**Figure 3. `<shps:InvocationContext>` — Schema Fragment**

An example `<shps:InvocationContext>` is shown below.

```
<shps:InvocationContext>
   <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
   <shps:InvokingPrincipal
            Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
            NameQualifier="http://idps-r-us.com" >
     uuid:23923-023843-230932-230923
   </shps:InvokingPrincipal>
   <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
</shps:InvocationContext>
```

**Example 2. Example `<shps:InvocationContext>`**

### 3.2.6. Callback EPR

The `<shps:CallbackEPR>` is used by the CSI to register its callback location for proxied service instances. This EPR is normally a traditional ID-WSF EPR (see [LibertyDisco]).

However, in the case where the CSI cannot expose an endpoint that is visible to the SHPS, the CSI should register an "anonymous" `<shps:CallbackEPR>` which MUST have the following characteristics:

- The ONLY element present in the EPR is the `<wsa:Address>` element which MUST have the value *http://www.w3.org/2005/08/addressing/anonymous*

The schema for the `<shps:CallbackEPR>` is shown below.

```
<!-- CallbackEPR - where the CSI can receive ProxyInvoke requests -->

<xs:element name="CallbackEPR" type="wsa:EndpointReferenceType"/>
```

**Figure 4. `<shps:CallbackEPR>` — Schema Fragment**

An example "anonymous" `<shps:CallbackEPR>` is shown below.

```
<shps:CallbackEPR>
  <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
</shps:CallbackEPR>
```

**Example 3. Example anonymous `<shps:CallbackEPR>`**

### 3.2.7. Service Descriptor

A Service Descriptor is used to describe a logical service that the SHPS is willing to host and/or proxy. The descriptor is an ID-WSF EPR (see [LibertyDisco]) that is further profiled as follows:

- The `<wsa:Address>` element MUST be *http://www.w3.org/2005/08/addressing/anonymous*
- A new element `<shps:ServiceMode>` is defined with the following schema:

```
<!-- ServiceMode - the proxied or hosted mode of the service instance -->

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

  This element MUST appear at least once within the `<wsa:Metadata>` element and MUST have one of the values described in Section 3.2.3 indicating the service mode(s) supported for this service by the SHPS.

- A new element <shps:ServiceStatus> is defined with the following schema:

```
<!-- ServiceStatus - the enabled/disabled status of the service instance -->

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

  If this Service Description describes a registered service instance at the SHPS, this element MUST appear exactly once and the element value MUST reflect the current status of the registered instance.

- A new element <shps:ServiceHandle> is defined with the following schema:

```
<!-- ServiceHandle - a reference to a hosted/proxied service instance -->

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

  If this Service Description describes a registered service instance at the SHPS, this element MUST appear exactly once and the element value MUST reflect the Service Handle assigned to the service by the SHPS.

- A new element <shps:CallbackEPR> (see Section 3.2.6) is defined to describe how the SHPS can communicate with the CSI to resolve proxied requests. The SHPS will send a <shps:ProxyInvoke> callback to this location whenever the SHPS receives an incoming request.

  The <shps:CallbackEPR> element MUST ONLY appear if the Service Descriptor is being used to register a proxied service and/or when describing a previously registered proxied service instance.

  If more than one <shps:CallbackEPR> element is present, the SHPS may choose any of the elements present (recognizing that the list is in preference order – the most preferred element is first).

- The <disco:SecurityContext> elements within the <wsa:Metadata> SHOULD NOT contain any security tokens. The <disco:SecurityContext> SHOULD only be used to describe the security mechanisms that the SHPS is willing to support for the service.

- The <disco:Abstract> and <disco:ProviderID> elements MAY be absent.

- A new sub-element <shps:ServiceHandle> MUST be present in the <wsa:Metadata> if the service description is describing a service instance registered at the SHPS (as opposed to just describing capabilities of the SHPS).

- The notOnOrAfter attribute MAY be used to indicate how long SHPS is willing to agree to host/proxy the service instance and/or how long the client would like the SHPS service to do so (depending upon the context – for registrations, it's the Advanced Client's desired time frame, for query responses, it's the SHPS desired time frame).

The complete Service Descriptor describes the service instance(s) that the SHPS is willing to host and/or proxy. The same Service Descriptor is used by the Advanced Client when registering it's desire. The Advanced Client will typically remove components of the Service Descriptor that it is not interested in utilizing (such as an older ServiceType definition the Advanced Client doesn't want exposed, or a security mechanism that doesn't meet the needs of the Advanced Client's CSI) and use this modified Service Descriptor to register at the SHPS.

The element/schema for the Service Descriptor is the actual WS-Addressing <wsa:EndpointReference> element. Thus there is no schema defined herein for the Service Descriptor.

An example <shps:ServiceDescriptor> is shown below.

```
<wsa:EndpointReference>
  <wsa:Address>
    http://www.w3.org/2005/08/addressing/anonymous
  </wsa:Address>

  <wsa:Metadata>
    <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
    <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
    <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
    <disco:ServiceType>urn:liberty:ps:2007-11</disco:ServiceType>
    <disco:Framework version="2.0" />
    <disco:SecurityContext>
      <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
      <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
    </disco:SecurityContext>
  </wsa:Metadata>
</wsa:EndpointReference>
```

**Example 4. Example `<wsa:EndpointReferences>` used as a Service Descriptor**

This example describes a hosted or proxied Liberty People Service (two "logical" versions of said People Service, one which doesn't really exist, but is used here to show how multiple logical versions would be described) and is able to support two security mechanisms within the ID-WSF 2.0 framework.

## 3.3. Operation: *Query*

The *Query* operation is used by the Advanced Client to obtain Service Descriptors from the SHPS which indicate which service(s) the SHPS is willing to host and/or proxy.

### 3.3.1. `wsa:Action` values for `Query` Messages

`<Query>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:Query`."

`<QueryResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:QueryResponse`."

### 3.3.2. `Query` Message

The `<Query>` is called to obtain a list of the Service Descriptors for the services the SHPS is willing to host and/or proxy for the Advanced Client.

The `<shps:Query>` request based on the Liberty Discovery Service's `<disco:Query>` request and is further profiled as follows:

- Zero or more `<shps:ServiceMode>` elements MAY be present within the `<disco:RequestedServiceType>` element. If present, this indicates the type of service mode requested by the client.

  Multiple `<shps:ServiceMode>` elements may be specified and any Service Descriptor that matches any of the specified `<shps:ServiceMode>` values is to be considered a match.

The interpretation of the request is the same interpretation of a request in the Discovery Service except that instead of searching against a set of SvcMD entries, the Advanced Client is searching against the set of Service Descriptors for the services available at the SHPS.

Similar to the Discovery Service's `<disco:Query>` an empty `<shps:Query>` element is treated as a request for all available Service Descriptors.

The schema for the `<shps:Query>` is shown below.

```
<!-- Query - query for ability to host or proxy services -->

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

**Figure 5. `<shps:Query>` — Schema Fragment**

An example message body containing a `<Query>` message follows. This request searches for a proxied instance of the Liberty ID-SIS Personal Profile Service that is exposed through the ID-WSF 2.0 framework using the TLS:SAML2 security mechanism.

```
<shps:Query>
  <disco:RequestedService>
    <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:sm:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:Framework version="2.0" />
    <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
  </disco:RequestedService>
</shps:Query>
```

**Example 5. Example `<shps:Query>` Message**

A second example message body containing a `<Query>` message follows. This is identical to the previous request except it is requesting information about a hosted service instance.

```
<shps:Query>
  <disco:RequestedService>
    <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
    <disco:SecurityMechID>urn:liberty:sm:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    <disco:Framework version="2.0" />
    <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
  </disco:RequestedService>
</shps:Query>
```

**Example 6. Example `<shps:Query>` Message**

### 3.3.3. `QueryResponse` Message

This response to the `<shps:Query>` request is similarly a profile of a Liberty Discovery Service data type, in this case the `disco:QueryResponseType`.

The elements in the response have the same definition and meaning as those documented for the `disco:QueryResponse`. with the exception that the ID-WSF EPRs returned in any successful response conform to the profile for Service Descriptors above.

```
  <!--QueryResponse - response for the Query request -->

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

**Figure 6. `<shps:QueryResponse>` — Schema Fragment**

An example `<shps:QueryResponse>` message showing a successful response with a proxied service instance description:

```
<shps:QueryResponse>
  <lu:Status code="OK" />
  <wsa:EndpointReference>
    <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    <wsa:Metadata>
      <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
      <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
      <disco:Framework version="2.0" />
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</shps:QueryResponse>
```

**Example 7. Example `<shps:QueryResponse>` Message**

An example `<shps:QueryResponse>` message showing a successful response with a hosted service instance description:

```
<shps:QueryResponse>
  <lu:Status code="OK" />
  <wsa:EndpointReference>
    <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    <wsa:Metadata>
      <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
      <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
      <disco:Framework version="2.0" />
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</shps:QueryResponse>
```

**Example 8. Example `<shps:QueryResponse>` Message**

### 3.3.4. Query Processing Rules

- The SHPS SHOULD return Service Descriptions for each service that it is willing to host or proxy which meets the conditions in the request.
- The SHPS MAY use any factor at its disposal in deciding whether it is willing to host or proxy a service including the identity of the user and/or the Advanced Client itself.
- 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 SHPS may not wish to reveal the reason for failure, in which case no second-level status code will appear.

## 3.4. Operation: *QueryRegistered*

The *QueryRegistered* operation is used by the Advanced Client to query for the Service Description of a service instance that the Advanced Client has registered.

This query should return the same data used to register the service description, including the `shps:CallbackEPR` – if any had been specified when it was registered.

### 3.4.1. `wsa:Action` values for `QueryRegistered` Messages

`<QueryRegistered>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:QueryRegistered`."

`<QueryRegisteredResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:QueryRegisteredResponse`."

### 3.4.2. `QueryRegistered` Message

The `<QueryRegistered>` request is called to retrieve the registered Service Description for a service instance at the SHPS.

The `<shps:QueryRegistered>` request contains one or more Service Handles for services instances previously registered at the SHPS.

The schema for the `<shps:QueryRegistered>` is shown below.

```
<!-- QueryRegistered - query for the registered service instances -->

<xs:element name="QueryRegistered" type="QueryRegisteredType"/>

<xs:complexType name="QueryRegisteredType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ServiceHandle" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 7. `<shps:QueryRegistered>` — Schema Fragment**

An example message body containing a `<QueryRegistered>` message follows. This request queries for two registered service instances.

```
<shps:QueryRegistered>
  <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
  <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
</shps:QueryRegistered>
```

**Example 9. Example `<shps:QueryRegistered>` Message**

### 3.4.3. `QueryRegisteredResponse` Message

This response to the <shps:QueryRegistered> request is a profile of a Liberty Discovery Service data type: the disco:QueryRegResponseType.

The elements in the response have the same definition and meaning as those documented for the disco:QueryRegResponse. with the exception that the ID-WSF EPRs returned in any successful response conform to the profile for Service Descriptors above.

```
<!-- QueryRegisteredResponse - response for QueryRegistered request -->

<xs:element name="QueryRegisteredResponse" type="disco:QueryResponseType"/>
```

**Figure 8. <shps:QueryRegResponse> — Schema Fragment**

```
<shps:QueryResponse>
  <lu:Status code="OK" />
  <wsa:EndpointReference>
    <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    <wsa:Metadata>
      <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
      <shps:ServiceHandle>uuid:23023-023802-2032023-023 8023</shps:ServiceHandle>
      <shps:ServiceStatus>urn:liberty:shps:2007-09:status:enabled</shps:ServiceStatus>
      <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
      <disco:Framework version="2.0" />
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</shps:QueryResponse>
```

**Example 10. Example <shps:QueryRegisteredResponse> Message**

Note that the ID-WSF EPR in the response includes both the <shps:ServiceMode> and <shps:ServiceHandle> elements. The Service Handle is used to match up the response data to the request data.

### 3.4.4. QueryRegistered Processing Rules

- The SHPS MUST only consider service instances registered by this Advanced Client in responding to the request. Service instances registered by other Advanced Clients MUST NOT be visible to this Advanced Client. There MUST be no difference in the response to a nonexistent Service Handle and a Service Handle that refers to a service instance registered by a different Advanced Client.

- The SHPS SHOULD return Service Descriptions for each registered service instance matching the specified Service Handle(s). If no Service Handles are specified, Service Descriptors for all of the registered service instances should be returned.

- If a specified Service Handle is not currently registered the SHPS SHOULD ignore it and only return the successful matches.

- If there were no successful matches of Service Handles, the top-level status code MUST be *Failed* and NO ID-WSF EPRs are to be returned. In such cases, if the SHPS is providing second level status codes, the second-level error code MUST be *NoResults*.

- If there are successful matches, the top-level status code MUST be "*OK*" and the successfully matching Service Descriptors are returned.

- The Advanced Client determines which service instances were found by examining the Service Handles within the returned Service Descriptions.

## 3.5. Operation: *Register*

The *Register* operation is used by the Advanced Client to request that the SHPS prepare to expose the specified service(s) on the Advanced Client's behalf.

Upon successful completion of this command the Advanced Client will have access to the service at the SHPS, but the service instance will **not** be enabled in the principal's DS resource. This allows the Advanced Client to register and initialize a service before it is made available through the principal's DS.

### 3.5.1. `wsa:Action` values for `Register` Messages

`<Register>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:Register`."

`<RegisterResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:RegisterResponse`."

### 3.5.2. `Register` Message

The `<Register>` request is called to register a new service to be exposed by the SHPS.

The `<shps:Register>` request contains one or more Service Descriptors for the services that the Advanced Client wants the SHPS to expose.

The schema for the `<shps:Register>` is shown below.

```
<!-- Register - request for a new hosted or proxied service instance -->

<xs:element name="Register" type="RegisterType"/>

<xs:complexType name="RegisterType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="wsa:EndpointReference" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 9. `<shps:Register>` — Schema Fragment**

An example message body containing a `<Register>` message follows. This request registers a hosted instance of the Liberty People Service that is exposed through the ID-WSF 2.0 framework using the TLS:SAML2 security mechanism.

```
<shps:Register>
  <wsa:EndpointReference lu:itemID="1">
    <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    <wsa:Metadata>
      <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
      <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
      <disco:Framework version="2.0" />
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
      </disco:SecurityContext>
    </wsa:Metadata>
  </wsa:EndpointReference>
</shps:Register>
```

**Example 11. Example `<shps:Register>` Message**

Another example message body containing a `<Register>` message follows. This request registers a proxied instance of the Liberty People Service that is exposed through the ID-WSF 2.0 framework using the TLS:SAML2 security mechanism. Note the CallbackEPR within the request specifying the anonymous endpoint indicating that the client will poll for requests.

```
<shps:Register>
  <wsa:EndpointReference lu:itemID="1">
    <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    <wsa:Metadata>
      <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
      <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
      <disco:Framework version="2.0" />
      <disco:SecurityContext>
        <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
      </disco:SecurityContext>
      <shps:CallbackEPR>
        <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      </shps:CallbackEPR>
    </wsa:Metadata>
  </wsa:EndpointReference>
</shps:Register>
```

**Example 12. Example `<shps:Register>` Message**

### 3.5.3. `RegisterResponse` Message

This response to the `<shps:Register>` request contains the following elements:

- `<lu:Status>` **[Required]** - The status of the response. See the processing rules below for more information.
- `<RegisterResponseItem>` **[Optional]** - Zero or more response items, one for each successful registration request. The content of each `<RegisterResponseItem>` is as follows:
  - `<shps:ServiceHandle>` **[Required]** - The service handle assigned to the service instance that was registered.
  - `ref` **[Required]** - The reference to the registration item in the request that this response is associated with.
- `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

The elements in the response have the same definition and meaning as those documented for the `disco:RegisterResponse` with the exception that the ID-WSF EPRs returned in any successful response conform to the profile for Service Descriptors above.

```
<!-- RegisterResponse - response to the Register request -->

<xs:element name="RegisterResponse" type="RegisterResponseType"/>

<xs:complexType name="RegisterResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="RegisterResponseItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="RegisterResponseItem" type="RegisterResponseItemType" />

<xs:complexType name="RegisterResponseItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required" />
</xs:complexType>
```

**Figure 10. `<shps:RegisterResponse>` — Schema Fragment**

An example message body containing a `<RegisterResponse>` message follows. This is a successful response.

```
<shps:RegisterResponse>
  <lu:Status code="OK" />
  <shps:RegisterResponseItem ref="1">
    <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
  </shps:RegisterResponseItem>
</shps:RegisterResponse>
```

**Example 13. Example `<shps:RegisterResponse>` Message**

An example message body containing a `<RegisterResponse>` message follows. This is a successful response for a registration of two service instances (hence the inclusion of two Service Handles in the response).

```
<shps:RegisterResponse>
  <lu:Status code="OK" />
  <shps:RegisterResponseItem ref="1">
    <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
  </shps:RegisterResponseItem>
  <shps:RegisterResponseItem ref="2">
    <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
  </shps:RegisterResponseItem>
</shps:RegisterResponse>
```

**Example 14. Example `<shps:RegisterResponse>` Message**

### 3.5.4. Register Processing Rules

- The <shps:Register> request MUST include an itemID attribute on each <wsa:EndpointReference> element in the request.

  The value of the itemID attribute MUST be placed into the ref attribute in the <shps:RegisterResponseItem> element associated with the <wsa:EndpointReference> element in the request.

- The order of the elements in the response MAY be different than the order of the associated elements in the request. The ref attribute MUST be used to correlate the response item to its associated request element.

- The Advanced Client MAY alter a Service Description to have more restrictive settings than those returned from the SHPS in a <shps:QueryResponse>. For example, the Advanced Client may select a single security mechanism for its service instance at the SHPS, even though the SHPS is capable of supporting several different security mechanisms.

  Any such alterations made SHOULD NOT include features or capabilities that the SHPSs has not expressed a willingness to expose.

- The SHPS MAY refuse to accept the registration of any Service Descriptor for which it is unable or unwilling to expose on behalf of the Advanced Client. In such cases the SHPS MUST return a failure.

- The Advanced Client MAY initiate a <shps:Register> request using a Service Descriptor that it created (rather than one obtained from the SHPS using the <shps:Query> interface). Of course, the SHPS MAY refuse such operations if it does not support exposing the requested service or any of the specific features requested.

- The SHPS MAY use any factor at its disposal in deciding whether or not to accept the registration (it MAY even deny a registration for a Service Descriptor that the SHPS previously returned to the Advanced Client in a <shps:QueryResponse> if the conditions for such willingness have changed).

- 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. This requirement is mostly applicable in the case where the request includes multiple Service Descriptors.

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

## 3.6. Operation: *Update*

The *Update* operation is used by the Advanced Client to update a service instance registration at the SHPS.

This interface is typically used by the Advanced Client to either update the <shps:CallbackEPR> for a proxied service or to update the service parameters for a hosted service (such as enabling support for additional security mechanisms that the SHPS is able to support, but were not chosen during the initial registration).

### 3.6.1. `wsa:Action` values for `Update` Messages

<Update> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:shps:2007-09:Update."

<UpdateResponse> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:shps:2007-09:UpdateResponse."

### 3.6.2. `Update` Message

The <Update> request is called to update the registered information for a service instance at the SHPS.

The `<shps:Update>` request contains one or more `<shps:UpdateItem>` elements which have the following contents:

- `<shps:ServiceHandle>`: **[Required]** - The Service Handle for the service instance that is being updated.
- `<wsa:EndpointReference>` **[required]** The updated Service Description. This is a complete replacement of the existing Service Description that was previously in place for the Service Instance.
- `itemID` **[required]** - The identifier for this update item (for correlation with the results).

The schema for the `<shps:Update>` is shown below.

```
<!-- Update - update the configuration of the hosted/proxied service instance -->

<xs:element name="Update" type="UpdateType"/>

<xs:complexType name="UpdateType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="UpdateItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="UpdateItem" type="UpdateItemType" />

<xs:complexType name="UpdateItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle"       />
    <xs:element ref="wsa:EndpointReference"  />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>

```

**Figure 11. `<shps:Update>` — Schema Fragment**

An example message body containing an `<Update>` message follows. This request updates two service instances at the SHPS service.

```
<shps:Update>
  <shps:UpdateItem itemID="1">
    <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
    <wsa:EndpointReference>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:Metadata>
        <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
        <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
        <disco:Framework version="2.0" />
        <disco:SecurityContext>
          <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
          <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:Bearer</disco:SecurityMechID>
        </disco:SecurityContext>
      </wsa:Metadata>
    </wsa:EndpointReference>
  </shps:UpdateItem>
  <shps:UpdateItem itemID="2">
    <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
    <wsa:EndpointReference>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:Metadata>
        <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
        <disco:ServiceType>urn:liberty:pp:2003-08</disco:ServiceType>
        <disco:Framework version="2.0" />
        <disco:SecurityContext>
          <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        </disco:SecurityContext>
      </wsa:Metadata>
    </wsa:EndpointReference>
  </shps:UpdateItem>
</shps:Update>
```

**Example 15. Example `<shps:Update>` Message**

### 3.6.3. `UpdateResponse` Message

This response to the `<shps:Update>` request contains the following elements/attributes:

- `<lu:Status>`: **[Required]** - The status of the response. See the processing rules below for more information.
- `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- UpdateResponse - the response to the Update request -->

<xs:element name="UpdateResponse" type="UpdateResponseType"/>

<xs:complexType name="UpdateResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>
```

**Figure 12. `<shps:UpdateResponse>` — Schema Fragment**

An example message body containing a `<UpdateResponse>` message follows. This is a partial success message for an attempted update of 2 service instances where the 1st succeeded and the 2nd failed.

```
<shps:UpdateResponse>
  <lu:Status code="Partial">
    <lu:Status code="NoSuchService" ref="2" />
  </lu:Status>
</shps:UpdateResponse>
```

**Example 16. Example `<shps:UpdateResponse>` Message**

### 3.6.4. Update Processing Rules

- If for any reason an update is not accepted, the existing service instance at the SHPS MUST NOT be affected.
- Service instances created by one Advanced Client MUST NOT be visible to other Advanced Clients. If an Advanced Client presents a Service Handle for a service instance that the Advanced Client did not create, the SHPS MUST behave as if that service instance did not exist and accordingly any attempt to update such a service should fail with an optional error code of "*NotFound*."
- The SHPS SHOULD refuse to make changes if they indicate features that are not supported by the SHPS instance (such as a security mechanism that the SHPS does not support). In such cases the SHPS MAY indicate this failure case by setting the second-level status code to: *FeatureNotSupported*.
- The SHPS MAY refuse to accept the update of any service instance for which it is unable or unwilling to expose on behalf of the Advanced Client. In such cases the SHPS MUST return a failure.
- The Advanced Client MAY initiate a `<shps:Update>` request using a Service Descriptor that it created (rather than one obtained from the SHPS using the `<shps:Query>` interface). Of course, the SHPS MAY refuse such operations if it does not support exposing the requested service or any of the specific features requested.
- The SHPS MAY use any factor at its disposal in deciding whether or not to accept the update (it MAY even deny an update with a Service Descriptor that the SHPS previously returned to the Advanced Client in a `<shps:QueryResponse>` if the conditions for such willingness has changed).
- Each `<shps:UpdateItem>` is processed independently and may independently succeed or fail on its own merits.
- If all `<shps:UpdateItem>` requests are successfully processed, the top-level status code MUST be "*OK*." If all of the items failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was not successful indicating so and including the `ref` attribute containing the `itemID` value for the item. These second-level status codes MAY simply be "*Failed*," or they may indicate with more detail the reason for the failure.
- If the top-level status code is "*Failed*," the response MAY also contain other status codes (such as *Forbidden*) as a second-level status code. The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

## 3.7. Operation: *Delete*

The *Delete* operation is used by the Advanced Client to delete a service instance registration at the SHPS.

Upon deletion the service will no longer be hosted or proxied by the SHPS. If the service instance was enabled (see Section 3.10), the service instance will also be disabled (e.g., calling delete will cause SHPS to remove the service instance from the principal's discovery service registration).

### 3.7.1. `wsa:Action` values for `Delete` Messages

`<Delete>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:Delete`."

`<DeleteResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:DeleteResponse`."

### 3.7.2. `Delete` Message

The `<Delete>` request is called to delete the registered service instance at the SHPS.

The `<shps:Delete>` request contains one or more `<shps:ServiceHandle>` elements (one for each service instance that is to be deleted).

The schema for the `<shps:Delete>` is shown below.

```
<!-- Delete - delete (remove) a service instance -->

<xs:element name="Delete" type="DeleteType"/>

<xs:complexType name="DeleteType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 13. `<shps:Delete>` — Schema Fragment**

An example message body containing an `<Delete>` message follows. This request deletes two service instances at the SHPS service.

```
<shps:Delete>
  <shps:ServiceHandle>uuid:23023-023802-203 2023-0238023</shps:ServiceHan dle>
  <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34E D3</shps:ServiceHandle>
</shps:Delete>
```

**Example 17. Example `<shps:Delete>` Message**

### 3.7.3. `DeleteResponse` Message

This response to the `<shps:Delete>` request contains the following elements/attributes:

- `<lu:Status>`: **[Required]** - The status of the response. See the processing rules below for more information.

- `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- DeleteResponse - response to a delete request -->

<xs:element name="DeleteResponse" type="DeleteResponseType"/>

<xs:complexType name="DeleteResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>
```

**Figure 14. `<shps:DeleteResponse>` — Schema Fragment**

An example message body containing a `<DeleteResponse>` message follows. This is a partial success message for an attempted delete of 2 service instances where the 1st succeeded and the 2nd failed.

```
<shps:DeleteResponse>
  <lu:Status code="Partial">
    <lu:Status code="NoSuchService"
            ref="uuid:97326-A726F2-9FE326A-8C34ED3" />
  </lu:Status>
</shps:DeleteResponse>
```

**Example 18. Example `<shps:DeleteResponse>` Message**

### 3.7.4. Delete Processing Rules

- If for any reason an delete is not accepted, the existing service instance at the SHPS MUST NOT be affected.

- Service instances created by one Advanced Client MUST NOT be visible to other Advanced Clients. If an Advanced Client presents a Service Handle for a service instance that the Advanced Client did not create, the SHPS MUST behave as if that service instance did not exist and accordingly any attempt to delete such a service should fail with an optional error code of "*NotFound*."

- Each `<shps:ServiceHandle>` is processed independently and may independently succeed or fail on its own merits.

- If all of the requested service handles are deleted, the top-level status code MUST be "*OK*." If all of the deletes failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was not successful indicating so and including the `ref` attribute containing the `ServiceHandle` value for the delete(s) that failed. These second-level status codes MAY simply be "*Failed*," or they may indicate with more detail the reason for the failure.

- If the top-level status code is "*Failed*," the response MAY also contain other status codes (such as *Forbidden*) as a second-level status code. The SHPS may not wish to reveal the reason for failure, in which case no second-level status code will appear.

## 3.8. Operation: *Invoke*

The *Invoke* operation is used by the Advanced Client to invoke the service interfaces on the SHPS using the Service Handle to identify the resource being targeted.

This interface is typically used by the Advanced Client to initialize and/or update information stored at the SHPS for hosted service instances.

### 3.8.1. `wsa:Action` values for `Invoke` Messages

`<Invoke>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:Invoke`."

`<InvokeResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:InvokeResponse`."

### 3.8.2. `Invoke` Message

The `<Invoke>` request is called to invoke a service interface for a service instance hosted by the SHPS.

The `<shps:Invoke>` request contains one or more `<shps:InvokeItem>` elements each of which have the following attributes and/or elements:

- `itemID`: **[Required]** - The `itemID` is a string that may have any value assigned by the requester (but MUST be different than the `itemID` assigned to any other `<shps:InvokeItem>` elements in the same request.

  The `itemID` is used to enable correlation of the results in the `<shps:InvokeResponse>` with the `<shps:InvokeItem>` that lead to those results.

- `<shps:ServiceHandle>` **[Required]** - The Service Handle for the service instance which this invocation item should apply to.

- `<xs:any>` **[Required]** - The service level request (any request is allowed by the service instance exposed by the SHPS). For example, if the Liberty People Service is the service that is being hosted by the SHPS, an `<ps:AddEntityRequest>` may be specified in this location.

  The individual service definitions (such as the Liberty People Service Specification) drive what is allowable within this location. Essentially, anything that MAY be placed into the body of a typical invocation of the service MAY be placed into the `<shps:InvokeItem>` element.

- `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

The schema for the <shps:Invoke> is shown below.

```
<!-- Invoke - invoke service level interface on hosted service instance -->

<xs:element name="Invoke" type="InvokeType"/>

<xs:complexType name="InvokeType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="InvokeItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- InvokeItem - container for each service level request -->

<xs:element name="InvokeItem" type="InvokeItemType"/>

<xs:complexType name="InvokeItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

**Figure 15. <shps:Invoke> — Schema Fragment**

An example message body containing a <Invoke> message follows. This request invokes the Liberty Personal Profile <pp:Modify> interface on the instance of the service hosted by the SHPS (and referenced by the specified <shps:ServiceHandle>).

```
<shps:Invoke>
  <shps:InvokeItem itemID="1">
    <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
    <pp:Modify>
      <pp:ModifyItem>
         ... modification data goes here ...
      </pp:ModifyItem>
    </pp:Modify>
  </shps:InvokeItem>
</shps:Invoke>
```

**Example 19. Example <shps:Invoke> Message**

### 3.8.3. InvokeResponse Message

This response to the <shps:Invoke> request contains the following elements:

- <lu:Status>: **[Required]** - The status of the response. See the processing rules below for more information.
- <shps:InvokeResponseItem>: **[Optional]** - The container element for the response data for an <shps:InvokeItem> which contains the following attributes/elements:

- `ref` **[Required]** The value from the `itemID` attribute on the `<shps:InvokeItem>` element in the request whose results are included in this `<shps:InvokeResponseItem>` element.

- `<xs:any>` **[Required]** - The service level response for the request. For example, if the Liberty People Service `<ps:AddEntityRequest>` had been specified in the request, the `<ps:AddEntityResponse>` would be placed here.

  The individual service definitions (such as the Liberty People Service Specification) drive what is expected within this location. Essentially, anything that MAY be placed into the body of a typical response of the service MAY be placed into the `<shps:InvokeResponseItem>` element.

- `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- InvokeResponse - response to an Invoke request -->

<xs:element name="InvokeResponse" type="InvokeResponseType"/>

<xs:complexType name="InvokeResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="InvokeResponseItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- InvokeResponseItem - container for the result of each InvokeItem -->

<xs:element name="InvokeResponseItem" type="InvokeResponseItemType"/>

<xs:complexType name="InvokeResponseItemType">
  <xs:sequence>
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required" />
</xs:complexType>
```

**Figure 16. `<shps:InvokeResponse>` — Schema Fragment**

An example message body containing a `<InvokeResponse>` message follows. This is a successful response for a Liberty Personal Profile modification request.

```
<shps:InvokeResponse>
  <lu:Status code="OK" />
  <shps:InvokeResponseItem ref="1">
    <pp:ModifyResponse>
        ... modification response data goes here ...
    </pp:ModifyResponse>
  </shps:InvokeResponseItem>
</shps:InvokeResponse>
```

**Example 20. Example `<shps:InvokeResponse>` Message**

### 3.8.4. Invoke Processing Rules

- The CSI MAY invoke any service interface necessary to populate the SHPS hosted service instance with sufficient data to properly respond to requests from WSCs.

- The SHPS MAY process the requested invocation items in any order it sees fit. CSIs SHOULD NOT include invocations of a service instance that are dependent upon each other as the results are not defined (the second invocation may take place prior to the first).

- Each `<shps:InvokeResponse>` element in the request MUST be treated independently.

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

## 3.9. Operation: *GetStatus*

The *GetStatus* operation is used by the Advanced Client to determine the status of a registered service instance at the SHPS.

### 3.9.1. `wsa:Action` values for `GetStatus` Messages

`<GetStatus>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:GetStatu`

`<GetStatusResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:GetStatusResponse`."

### 3.9.2. `GetStatus` Message

The `<GetStatus>` request is called to get the current status of the specified service instance at the SHPS.

The `<shps:GetStatus>` request contains the following attributes and/or elements:

- `<shps:ServiceHandle>` **[Optional]** Zero or more Service Handles for the service instances who's status is requested. If not specified, the status of all service instances registered by the calling Advanced Client should be returned.

- `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

The schema for the `<shps:GetStatus>` is shown below.

```
<!-- GetStatus - Get the status of a Service Instance at SHPS -->

<xs:element name="GetStatus" type="GetStatusType"/>

<xs:complexType name="GetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

**Figure 17. `<shps:GetStatus>` — Schema Fragment**

An example message body containing a `<GetStatus>` message follows.

```
<shps:GetStatus>
  <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
  <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
</shps:GetStatus>
```

**Example 21. Example `<shps:GetStatus>` Message**

### 3.9.3. `GetStatusResponse` Message

This response to the `<shps:GetStatus>` request contains the following elements and/or attributes:

- `<lu:Status>`: **[Required]** - the status of the response. See the processing rules below for more information.
- `<shps:GetStatusResponseItem>` : **[Optional]** - zero or more Service Handle/Service Status pairs indicating the current status of the specified service. This element MUST be absent if there were no service instances registered at the SHPS that met the conditions of the query or if the call otherwise failed.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- GetStatusResponse - response to the GetStatus request -->

<xs:element name="GetStatusResponse" type="GetStatusResponseType"/>

<xs:complexType name="GetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="GetStatusResponseItem" minOccurs="0"
                                    maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- GetStatusResponseItem - the status of a singe service instance -->

<xs:element name="GetStatusResponseItem" type="GetStatusResponseItemType"/>

<xs:complexType name="GetStatusResponseItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:element ref="ServiceStatus" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

**Figure 18. `<shps:GetStatusResponse>` — Schema Fragment**

An example message body containing a `<GetStatusResponse>` message follows. This is a successful response showing the status of two service instances.

```
<shps:GetStatusResponse>
  <lu:Status code="OK" />
  <shps:GetStatusResponseItem>
    <shps:ServiceHandle>uuid:2302 3-023802-2032023-0238023</shps :ServiceHandle>
    <shps:ServiceStatus>urn:liberty:shps :2007-09:status:enabled</sh ps:ServiceStatus>
  </shps:GetStatusResponseItem>
  <shps:GetStatusResponseItem>
    <shps:ServiceHandle>uuid:97326 -A726F2-9FE326A-8C34ED3</shps: ServiceHandle>
    <shps:ServiceStatus>urn:liberty:shps: 2007-09:status:enabled</shp s:ServiceStatus>
  </shps:GetStatusResponseItem>
</shps:GetStatusResponse>
```

**Example 22. Example `<shps:GetStatusResponse>` Message**

### 3.9.4. GetStatus Processing Rules

- If there is no `<ServiceHandle>` on the request, the SHPS MUST interpret this request as a request for the status of ALL services instances registered at the SHPS by the invoking Advanced Client.

- A request for a service instance which is NOT registered for the Advanced Client (whether or not it is registered for another Advanced Client) is, for the SHPS, considered successful and simply does not result in a `<GetStatusResponseItem>` in the results of the operation. For example, if the `<GetStatus>` included a single `<ServiceHandle>` referring to a service instance not currently registered at the SHPS, the response would be:

```
<shps:GetStatusResponse>
  <lu:Status code="OK" />
</shps:GetStatusResponse>
```

- If there are any matching results (even if there are some other non-matching elements), the request is successful and 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 *NoResults* as a second-level status code. The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

## 3.10. Operation: *SetStatus*

The *SetStatus* operation is used by the Advanced Client to enable or disable a registered service instance at the SHPS.

This interface is typically used by the Advanced Client after it has completed the initialization of the hosted/proxied service to make the service visible to WSCs.

The primary effect of this call is that the SHPS registers and associates (or disassociates) the hosted or proxied service instance in the principal's DS resource. This makes the service instance visible (or invisible) to WSCs.

### 3.10.1. `wsa:Action` values for `SetStatus` Messages

`<SetStatus>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:SetStatu`

`<SetStatusResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:SetStatusResponse`."

### 3.10.2. `SetStatus` Message

The `<SetStatus>` request is called to set the current status of the specified service instance to the specified status.

The `<shps:SetStatus>` request contains the following attributes and/or elements:

- `SetStatusItem`: **[Required]** - one or more set status items each of which contain:
  - `ServiceStatus`: **[Required]** - the new status for the specified service instance(s). This MUST be set to one of the following values:
    - *urn:liberty:shps:2007-09:status:enabled*

      The service instance is to be made visible in the principal's DS resource such that WSCs can discover and invoke it.
    - *urn:liberty:shps:2007-09:status:disabled*

      The service instance is to be removed from the principal's DS resource such that WSCs can no longer discover and invoke it.
  - `<shps:ServiceHandle>` **[Required]** - one or more Service Handles for the service instances which are to be enabled or disabled.

- `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

The schema for the `<shps:SetStatus>` is shown below.

```
<!-- SetStatus - set the status of a hosted/proxied service instance -->

<xs:element name="SetStatus" type="SetStatusType"/>

<xs:complexType name="SetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="SetStatusItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="SetStatusItem" type="SetStatusItemType"/>

<xs:complexType name="SetStatusItemType">
  <xs:sequence>
    <xs:element ref="ServiceStatus" />
    <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>
```

**Figure 19. `<shps:SetStatus>` — Schema Fragment**

An example message body containing a `<SetStatus>` message follows. This request enables two service instances.

```
<shps:SetStatus>
  <shps:SetStatusItem itemID="1" >
    <shps:ServiceStatus>urn:liberty:shps:2007-09:status:disabled</shps:ServiceStatus>
    <shps:ServiceHandle>uuid:53723-123872-5223223-8237823</shps:ServiceHandle>
  </shps:SetStatusItem>
  <shps:SetStatusItem itemID="2">
    <shps:ServiceStatus>urn:liberty:shps:2007-09:status:enabled</shps:ServiceStatus>
    <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
    <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
  </shps:SetStatusItem>
</shps:SetStatus>
```

**Example 23. Example `<shps:SetStatus>` Message**

### 3.10.3. `SetStatusResponse` Message

This response to the `<shps:SetStatus>` request contains the following elements:

- `<lu:Status>`: **[Required]** - The status of the response. See the processing rules below for more information.
- `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- SetStatusResponse - response to the SetStatus request -->

<xs:element name="SetStatusResponse" type="SetStatusResponseType"/>

<xs:complexType name="SetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>
```

**Figure 20. `<shps:SetStatusResponse>` — Schema Fragment**

An example message body containing a `<SetStatusResponse>` message follows. This is a successful response.

```
<shps:SetStatusResponse>
  <lu:Status code="OK" />
</shps:SetStatusResponse>
```

**Example 24. Example `<shps:SetStatusResponse>` Message**

### 3.10.4. SetStatus Processing Rules

- 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. This requirement is mostly applicable in the case where the request includes multiple Service Descriptors.

- Each `<shps:SetStatusItem>` MUST be processed independently (even if one fails, the other `<shps:SetStatusItem>` elements MUST be processed.

- If request processing succeeded, the top-level status code MUST be "*OK*." Otherwise, the top-level status code MUST be "*Failed*" If all `<shps:SetStatusItem>` requests are successfully processed, the top-level status code MUST be "*OK*." If all of the items failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was not successful indicating so and including the `ref` attribute containing the `itemID` value for the item. These second-level status codes MAY simply be "*Failed*," or they may indicate with more detail the reason for the failure.

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

## 3.11. Operation: *Poll*

The *Poll* operation is used by the CSI to poll for any new requests when the CSI is unable to expose an externally visible endpoint for an incoming `shps:ProxyInvoke` interface for direct access by the SHPS.

This operation is an adaptation of the *Poll* design pattern and inherits all of its structure and processing rules.

### 3.11.1. `wsa:Action` values for `Poll` Messages

`<Poll>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:Poll`."

`<PollResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:PollResponse`."

### 3.11.2. `Poll` Message

The `<Poll>` request is called by the CSI to ask the SHPS for any queued `shps:ProxyInvoke` requests and to return any `shps:ProxyInvokeResponse` responses to prior requests.

The structure of the `<shps:Poll>` request is derived from the structure of the `<dp:PollType>` without modification. See [LibertyDP] for a complete description of the structure and meaning of the elements.

The schema for the `<shps:Poll>` is shown below.

```
<!-- Poll - Poll for new service requests -->

<xs:element name="Poll" type="dp:PollType"/>

```

**Figure 21. `<shps:Poll>` — Schema Fragment**

An example message body containing a `<shps:Poll>` message follows. This is a request asking for `shps:ProxyInvoke` requests and asks SHPS to wait for 5 minutes if none are immediately available.

```
<shps:Poll wait="300">
  <wsa:Action>urn:liberty:shps:2007-09:ProxyInvoke</wsa:Action>
</shps:Poll>

```

**Example 25. Example `<shps:Poll>` Message**

Another example message body containing a `<shps:Poll>` message follows. This example also includes a `shps:ProxyInvokeResponse` from a prior request received at the CSI.

```
<shps:Poll wait="300">
  <wsa:Action>urn:liberty:shps:2007-09:ProxyInvoke</wsa:Action>
  <dp:Response ref="1">
    <shps:ProxyInvokeResponse>
      <lu:Status code="OK" />
      <shps:ProxyInvokeResponseItem ref="1">
        <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
        <shps:ResponseHeaders>
          <sb:UsageDirectives> .....  </sb:UsageDirectives>
        </shps:ResponseHeaders>
        <pp:QueryResponse>
            ... modification response data goes here ...
        </pp:QueryResponse>
      </shps:ProxyInvokeResponseItem>
    </shps:ProxyInvokeResponse>
  </dp:Response>
</shps:Poll>
```

**Example 26. Example `<shps:Poll>` Message with a `shps:ProxyInvokeResponse`.**

### 3.11.3. `PollResponse` Message

This response to the `<shps:Poll>` request is derived from the `<dp:PollResponseType>` without modification. See [LibertyDP] for a complete description of the structure and meaning of the elements.

```
<!--PollResponse - response for the Poll request -->

<xs:element name="PollResponse" type="dp:PollResponseType"/>
```

**Figure 22. `<shps:PollResponse>` — Schema Fragment**

An example message body containing a `<shps:PollResponse>` message follows. This is a successful response without an embedded `shps:ProxyInvoke` request (and therefore there were no queued requests) and SHPS is advising the CSI to poll again in 10 minutes (600 seconds).

```
<shps:PollResponse nextPoll="600">
  <lu:Status code="OK" />
</shps:PollResponse>
```

**Example 27. Example `<shps:PollResponse>` Message**

Another example message body containing a `<shps:PollResponse>` message follows. This is a successful response with an embedded `shps:ProxyInvoke` request.

```
<shps:PollResponse>
  <lu:Status code="OK" />
  <dp:Request itemID="1">
    <shps:ProxyInvoke>
      <shps:ProxyInvokeItem itemID="1">
        <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
        <shps:InvocationContext>
          <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
          <shps:InvokingPrincipal
                 Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
                 NameQualifier="http://idps-r-us.com" >
            uuid:23923-023843-230932-230923
          </shps:InvokingPrincipal>
          <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
        </shps:InvocationContext>
        <shps:RequestHeaders>
          <sb:ProcessingContext>
            urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
          </sb:ProcessingContext>
        </shps:RequestHeaders>
        <pp:Query>
          ... query data goes here ...
        </pp:Query>
      </shps:ProxyInvokeItem>
    </shps:ProxyInvoke>
  </dp:Request>
</shps:PollResponse>
```

**Example 28. Example `<shps:PollResponse>` Message**

### 3.11.4. Poll Processing Rules

- All of the processing rules defined for the *Poll* design pattern MUST be followed. See [LibertyDP] for further information.

## 3.12. Operation: *ProxyInvoke*

The *ProxyInvoke* operation is NOT exposed by the SHPS service, but rather MUST be exposed by an Advanced Client that uses the proxying capability of SHPS. This interface is what SHPS calls when it has been invoked in order to get the data to use in the response.

The Advanced Client must either expose this interface directly via an externally visible endpoint (specified in the `<shps:CallbackEPR>` – see Section 3.2.6) or MUST use the `<shps:Poll>` interface to poll for invocations.

### 3.12.1. `wsa:Action` values for `ProxyInvoke` Messages

`<ProxyInvoke>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:ProxyInvoke`."

`<ProxyInvokeResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:shps:2007-09:ProxyInvokeResponse`."

### 3.12.2. `ProxyInvoke` Message

The `<ProxyInvoke>` request is called by the SHPS to pass on an invocation received from a WSC so that the Advanced Client can process the request and send the results to the SHPS who can then respond to the WSC.

The `<shps:ProxyInvoke>` request contains one or more `<shps:ProxyInvokeItem>` elements each of which have the following attributes and/or elements:

- `itemID`: **[Required]** - a string that may have any value assigned by the requester (but MUST be different than the `itemID` assigned to any other `<shps:ProxyInvokeItem>` elements in the same request.

  The `itemID` is used to enable correlation of the results in the `<shps:ProxyInvokeResponse>` with the `<shps:ProxyInvokeItem>` that lead to those results.

- `<shps:ServiceHandle>` **[Required]** - the Service Handle for the service instance which this `<shps:ProxyInvokeItem>` applies to. Different Service Handles MAY be specified in different `<shps:ProxyInvokeItem>` elements in the same request if the Advanced Client used the same `<shps:CallbackEPR>` for multiple service instances and multiple simultaneous invocations took place at the SHPS.

- `<shps:InvocationContext>` **[Required]** - the invocation context (see Section 3.2.5 of the service instance at the SHPS (necessary information for the CSI to make appropriate decisions about the results of the operation).

- `<shps:RequestHeaders>` **[Optional]** - a container for zero or more SOAP headers from the call received by the SHPS which the SHPS considers necessary for the CSI to resolve the request. Typical candidate headers include (but are **not** limited to):

  - `<sb:ApplicationEPR>`
  - `<sb:Consent>`
  - `<sb:ProcessingContext>`
  - `<sb:Timeout>`
  - `<sb:UsageDirectives>`

- `<xs:any>` **[Required]** - the unmodified incoming request that the SHPS received from the WSC. For example, if the Liberty Profile Service is the service that is being proxied by the SHPS, an `<pp:Query>` may be specified in this location.

  The individual service definitions (such as the Liberty Personal Profile Service Specification) drive what is allowable within this location. Essentially, anything that MAY be placed into the body of a typical invocation of the service MAY be placed into the `<shps:ProxyInvokeItem>` element.

- `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

The schema for the <shps:ProxyInvoke> is shown below.

```
<!-- ProxyInvoke - proxied invocation of CSI -->

<xs:element name="ProxyInvoke" type="ProxyInvokeType"/>

<xs:complexType name="ProxyInvokeType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ProxyInvokeItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- Declaration of ProxyInvokeItem element -->
<xs:element name="ProxyInvokeItem" type="ProxyInvokeItemType"/>

<xs:complexType name="ProxyInvokeItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:element ref="InvocationContext" />
    <xs:element ref="RequestHeaders" minOccurs="0" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="RequestHeaders" />
```

**Figure 23. <shps:ProxyInvoke> — Schema Fragment**

An example message body containing a <ProxyInvoke> message follows. This is a proxied Liberty Personal Profile <pp:Query> request on the service instance referenced by the specified <shps:ServiceHandle>).

```
<shps:ProxyInvoke>
  <shps:ProxyInvokeItem itemID="1">
    <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
    <shps:InvocationContext>
      <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
      <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
    </shps:InvocationContext>
    <pp:Query>
      ... query data goes here ...
    </pp:Query>
  </shps:ProxyInvokeItem>
</shps:ProxyInvoke>
```

**Example 29. Example <shps:ProxyInvoke> Message**

### 3.12.3. ProxyInvokeResponse Message

This response to the <shps:ProxyInvoke> request contains the following elements:

- `<lu:Status>`: **[Required]** The status of the response. See the processing rules below for more information.
- `<shps:ProxyInvokeResponseItem>` : **[Optional]** The container element for the response data for an `<shps:ProxyInvokeItem>` which contains the following attributes/elements:
  - `ref`: **[Required]** - the value from the `itemID` attribute on the `<shps:ProxyInvokeItem>` element in the request whose results are included in this `<shps:ProxyInvokeResponseItem>` element.
  - `<shps:ServiceHandle>` **[Required]** - the Service Handle for the service instance which this `<shps:ProxyInvokeResponseItem>` applies to.
  - `<shps:ResponseHeaders>` **[Optional]** - a container for zero or more SOAP headers to be included on the SHPS service response. Typical candidate headers include (but are **not** limited to):
    - `<sb:CredentialsContext>`
    - `<sb:UsageDirectives>`
    - `<sb:UserInteraction>`
  - `<xs:any>` **[Required]** - the service level response for the request. For example, if the Liberty People Service `<ps:AddEntityRequest>` had been specified in the request, the `<ps:AddEntityResponse>` would be placed here.

    The individual service definitions (such as the Liberty People Service Specification) drive what is expected within this location. Essentially, anything that MAY be placed into the body of a typical response of the service MAY be placed into the `<shps:ProxyInvokeResponseItem>` element.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- ProxyInvokeResponse - response to the ProxyInvoke request -->

<xs:element name="ProxyInvokeResponse" type="ProxyInvokeResponseType"/>

<xs:complexType name="ProxyInvokeResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="ProxyInvokeResponseItem" minOccurs="0"
                                    maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ProxyInvokeResponseItem - container for each service level invocation -->

<xs:element name="ProxyInvokeResponseItem" type="ProxyInvokeResponseItemType"/>

<xs:complexType name="ProxyInvokeResponseItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:element ref="ResponseHeaders" minOccurs="0" />
    <xs:any namespace="##other"
             processContents="lax"
             minOccurs="0"
             maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required" />
</xs:complexType>

<xs:element name="ResponseHeaders" />
```

**Figure 24. `<shps:ProxyInvokeResponse>` — Schema Fragment**

An example message body containing a `<ProxyInvokeResponse>` message follows. This is a successful response for a Liberty Personal Profile query request.

```
<shps:ProxyInvokeResponse>
  <lu:Status code="OK" />
  <shps:ProxyInvokeResponseItem ref="1">
    <shps:ServiceHandle>uuid:23023-023802-2032023-0 238023</shps:ServiceHandle>
    <shps:ResponseHeaders>
      <sb:UsageDirectives> .....  </sb:UsageDirectives>
    </shps:ResponseHeaders>
    <pp:QueryResponse>
        ... query response data goes here ...
    </pp:QueryResponse>
  </shps:ProxyInvokeResponseItem>
</shps:ProxyInvokeResponse>
```

**Example 30. Example `<shps:ProxyInvokeResponse>` Message**

### 3.12.4. ProxyInvoke Processing Rules

- The SHPS MUST build the `<shps:InvocationContext>` from the message it received from the WSC.
- Each `<shps:ProxyInvokeItem>` MUST be processed independently (even if one fails, the other `<shps:ProxyInvokeItem>` elements MUST be processed.
- Each `<shps:ProxyInvokeItem>` MAY address different service instances by including different service handles (if the CSI registered the same `<shps:CallbackEPR>` for multiple service instances.
- If all `<shps:ProxyInvokeItem>` requests are successfully processed, the top-level status code MUST be "*OK*." If all of the items failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was not successful indicating so and including the `ref` attribute containing the `itemID` value for the item. These second-level status codes MAY simply be "*Failed*," or they may indicate with more detail the reason for the failure.

  Note that the status is of the processing of the `<shps:ProxyInvokeItem>` elements themselves, not the service level processing (which would be reflected within the service response), So a `<shps:ProxyInvokeResponse>` message MAY have a status code of "*OK*" while some service level responses may include status codes of "*Failed*" (since the CSI successfully processed the ProxyRequest and is returning a failure service response).
- If the top-level status code is "*Failed*," the response MAY also contain other status codes (such as *Forbidden*) as a second-level status code. The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

# 4. Service Hosting/Proxying Service Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:shps:2007-09"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:disco="urn:liberty:disco:2006-08"
   xmlns:dp="urn:liberty:dp:2007-09"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"

   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

   xmlns="urn:liberty:shps:2007-09"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>

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

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

  <xs:import namespace="urn:liberty:dp:2007-09"
    schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>

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

  <xs:import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
    schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd" />

  <!-- Data Definitions { -->

  <!-- ServiceHandle - a reference to a hosted/proxied service instance -->

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


  <!-- ServiceMode - the proxied or hosted mode of the service instance -->

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


  <!-- ServiceStatus - the enabled/disabled status of the service instance -->

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


  <!-- InvocationContext - how the proxied service instance was invoked -->

  <xs:element name="InvocationContext" type="InvocationContextType" />

  <xs:complexType name="InvocationContextType">
    <xs:sequence>
      <xs:element ref="InvokingProvider" />
      <xs:element ref="InvokingPrincipal" minOccurs="0" />
      <xs:element ref="disco:SecurityMechID" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:element name="InvokingProvider"  type="xs:string" />
  <xs:element name="InvokingPrincipal" type="saml2:NameIDType" />

```

```
<!-- CallbackEPR - where the CSI can receive ProxyInvoke requests -->

<xs:element name="CallbackEPR" type="wsa:EndpointReferenceType"/>



<!-- End of  Data Definitions } -->

<!-- Interface Definitions { -->



<!-- RequestAbstractType - common request message structure -->

<xs:complexType name="RequestAbstractType" abstract="true">
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


<!-- ResponseAbstractType - common message response structure -->

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


<!-- Delete - delete (remove) a service instance -->

<xs:element name="Delete" type="DeleteType"/>

<xs:complexType name="DeleteType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- DeleteResponse - response to a delete request -->

<xs:element name="DeleteResponse" type="DeleteResponseType"/>

<xs:complexType name="DeleteResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>


<!-- GetStatus - Get the status of a Service Instance at SHPS -->

<xs:element name="GetStatus" type="GetStatusType"/>

<xs:complexType name="GetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>


<!-- GetStatusResponse - response to the GetStatus request -->

<xs:element name="GetStatusResponse" type="GetStatusResponseType"/>

<xs:complexType name="GetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="GetStatusResponseItem" minOccurs="0"
                                    maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- GetStatusResponseItem - the status of a singe service instance -->

<xs:element name="GetStatusResponseItem" type="GetStatusResponseItemType"/>

<xs:complexType name="GetStatusResponseItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:element ref="ServiceStatus" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>



<!-- Query - query for ability to host or proxy services -->

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


<!--QueryResponse - response for the Query request -->

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


<!-- Invoke - invoke service level interface on hosted service instance -->

<xs:element name="Invoke" type="InvokeType"/>

<xs:complexType name="InvokeType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="InvokeItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- InvokeItem - container for each service level request -->

<xs:element name="InvokeItem" type="InvokeItemType"/>

<xs:complexType name="InvokeItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
```

```
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>


<!-- InvokeResponse - response to an Invoke request -->

<xs:element name="InvokeResponse" type="InvokeResponseType"/>

<xs:complexType name="InvokeResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="InvokeResponseItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- InvokeResponseItem - container for the result of each InvokeItem -->

<xs:element name="InvokeResponseItem" type="InvokeResponseItemType"/>

<xs:complexType name="InvokeResponseItemType">
  <xs:sequence>
    <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required" />
</xs:complexType>


<!-- QueryRegistered - query for the registered service instances -->

<xs:element name="QueryRegistered" type="QueryRegisteredType"/>

<xs:complexType name="QueryRegisteredType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ServiceHandle" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- QueryRegisteredResponse - response for QueryRegistered request -->

<xs:element name="QueryRegisteredResponse" type="disco:QueryResponseType"/>


<!-- Register - request for a new hosted or proxied service instance -->

<xs:element name="Register" type="RegisterType"/>

<xs:complexType name="RegisterType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="wsa:EndpointReference" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>


<!-- RegisterResponse - response to the Register request -->

<xs:element name="RegisterResponse" type="RegisterResponseType"/>

<xs:complexType name="RegisterResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="RegisterResponseItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="RegisterResponseItem" type="RegisterResponseItemType" />

<xs:complexType name="RegisterResponseItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required" />
</xs:complexType>


<!-- SetStatus - set the status of a hosted/proxied service instance -->

<xs:element name="SetStatus" type="SetStatusType"/>

<xs:complexType name="SetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="SetStatusItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="SetStatusItem" type="SetStatusItemType"/>

<xs:complexType name="SetStatusItemType">
  <xs:sequence>
    <xs:element ref="ServiceStatus" />
    <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>


<!-- SetStatusResponse - response to the SetStatus request -->

<xs:element name="SetStatusResponse" type="SetStatusResponseType"/>

<xs:complexType name="SetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>


<!-- Update - update the configuration of the hosted/proxied service instance -->

<xs:element name="Update" type="UpdateType"/>
```

```
<xs:complexType name="UpdateType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="UpdateItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="UpdateItem" type="UpdateItemType" />

<xs:complexType name="UpdateItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle"       />
    <xs:element ref="wsa:EndpointReference"  />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>


<!-- UpdateResponse - the response to the Update request -->

<xs:element name="UpdateResponse" type="UpdateResponseType"/>

<xs:complexType name="UpdateResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>


<!-- Poll - Poll for new service requests -->

<xs:element name="Poll" type="dp:PollType"/>


<!--PollResponse - response for the Poll request -->

<xs:element name="PollResponse" type="dp:PollResponseType"/>


<!-- ProxyInvoke - proxied invocation of CSI -->

<xs:element name="ProxyInvoke" type="ProxyInvokeType"/>

<xs:complexType name="ProxyInvokeType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ProxyInvokeItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- Declaration of ProxyInvokeItem element -->
<xs:element name="ProxyInvokeItem" type="ProxyInvokeItemType"/>

<xs:complexType name="ProxyInvokeItemType">
  <xs:sequence>
    <xs:element ref="ServiceHandle" />
    <xs:element ref="InvocationContext" />
    <xs:element ref="RequestHeaders" minOccurs="0" />
    <xs:any namespace="##other"
```

```
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="itemID" type="xs:string" use="required" />
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:element name="RequestHeaders" />


  <!-- ProxyInvokeResponse - response to the ProxyInvoke request -->

  <xs:element name="ProxyInvokeResponse" type="ProxyInvokeResponseType"/>

  <xs:complexType name="ProxyInvokeResponseType">
    <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
        <xs:sequence>
          <xs:element ref="ProxyInvokeResponseItem" minOccurs="0"
                                      maxOccurs="unbounded" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- ProxyInvokeResponseItem - container for each service level invocation -->

  <xs:element name="ProxyInvokeResponseItem" type="ProxyInvokeResponseItemType"/>

  <xs:complexType name="ProxyInvokeResponseItemType">
    <xs:sequence>
      <xs:element ref="ServiceHandle" />
      <xs:element ref="ResponseHeaders" minOccurs="0" />
      <xs:any namespace="##other"
              processContents="lax"
              minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ref" type="xs:string" use="required" />
  </xs:complexType>

  <xs:element name="ResponseHeaders" />



  <!-- End of Interface Definitions } -->

</xs:schema>
```

# 5. Service Hosting/Proxying Service WSDL

```
<?xml version="1.0"?>
<definitions name="shps-svc"
  targetNamespace="urn:liberty:shps:2007-09"
  xmlns:tns="urn:liberty:shps:2007-09"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
  xmlns:shps="urn:liberty:shps:2007-09"
  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:shps:2007-09"
               schemaLocation="liberty-idwsf-shps-v1.0.xsd" />
    </xsd:schema>
  </types>

  <message name="Query">
    <part name="body" element="shps:Query"/>
  </message>
  <message name="QueryResponse">
    <part name="body" element="shps:QueryResponse"/>
  </message>

  <message name="QueryRegistered">
    <part name="body" element="shps:QueryRegistered"/>
  </message>
  <message name="QueryRegisteredResponse">
    <part name="body" element="shps:QueryRegisteredResponse"/>
  </message>

  <message name="Register">
    <part name="body" element="shps:Register"/>
  </message>
  <message name="RegisterResponse">
    <part name="body" element="shps:RegisterResponse"/>
  </message>

  <message name="Update">
    <part name="body" element="shps:Update"/>
  </message>
  <message name="UpdateResponse">
    <part name="body" element="shps:UpdateResponse"/>
  </message>

  <message name="Delete">
    <part name="body" element="shps:Delete"/>
  </message>
  <message name="DeleteResponse">
    <part name="body" element="shps:DeleteResponse"/>
  </message>

  <message name="Invoke">
    <part name="body" element="shps:Invoke"/>
  </message>
  <message name="InvokeResponse">
    <part name="body" element="shps:InvokeResponse"/>
  </message>

  <message name="GetStatus">
```

```
  <part name="body" element="shps:GetStatus"/>
 </message>
 <message name="GetStatusResponse">
  <part name="body" element="shps:GetStatusResponse"/>
 </message>

 <message name="SetStatus">
  <part name="body" element="shps:SetStatus"/>
 </message>
 <message name="SetStatusResponse">
  <part name="body" element="shps:SetStatusResponse"/>
 </message>

 <portType name="SHPSPort">

  <operation name="Query">
   <input message="tns:Query"
     wsaw:Action="urn:liberty:shps:2007-09:Query" />
   <output message="tns:QueryResponse"
    wsaw:Action="urn:liberty:shps:2007-09:QueryResponse" />
  </operation>

  <operation name="QueryRegistered">
   <input message="tns:QueryRegistered"
     wsaw:Action="urn:liberty:shps:2007-09:QueryRegistered" />
   <output message="tns:QueryRegisteredResponse"
    wsaw:Action="urn:liberty:shps:2007-09:QueryRegisteredResponse" />
  </operation>

  <operation name="Register">
   <input message="tns:Register"
     wsaw:Action="urn:liberty:shps:2007-09:Register" />
   <output message="tns:RegisterResponse"
    wsaw:Action="urn:liberty:shps:2007-09:RegisterResponse" />
  </operation>

  <operation name="Update">
   <input message="tns:Update"
     wsaw:Action="urn:liberty:shps:2007-09:Update" />
   <output message="tns:UpdateResponse"
    wsaw:Action="urn:liberty:shps:2007-09:UpdateResponse" />
  </operation>

  <operation name="Delete">
   <input message="tns:Delete"
     wsaw:Action="urn:liberty:shps:2007-09:Delete" />
   <output message="tns:DeleteResponse"
    wsaw:Action="urn:liberty:shps:2007-09:DeleteResponse" />
  </operation>

  <operation name="Invoke">
   <input message="tns:Invoke"
     wsaw:Action="urn:liberty:shps:2007-09:Invoke" />
   <output message="tns:InvokeResponse"
    wsaw:Action="urn:liberty:shps:2007-09:InvokeResponse" />
  </operation>

  <operation name="GetStatus">
   <input message="tns:GetStatus"
     wsaw:Action="urn:liberty:shps:2007-09:GetStatus" />
   <output message="tns:GetStatusResponse"
    wsaw:Action="urn:liberty:shps:2007-09:GetStatusResponse" />
  </operation>

  <operation name="SetStatus">
   <input message="tns:SetStatus"
     wsaw:Action="urn:liberty:shps:2007-09:SetStatus" />
```

```
    <output message="tns:SetStatusResponse"
     wsaw:Action="urn:liberty:shps:2007-09:SetStatusResponse" />
  </operation>

</portType>

<!--
An example of a binding and service that can be used with this
abstract service description is shpsided below.
-->

<binding name="SHPSBinding" type="tns:SHPSPort">

  <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>

  <operation name="Query">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="QueryRegistered">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="Register">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="Update">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="Delete">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="Invoke">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="GetStatus">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

  <operation name="SetStatus">
   <input> <soap:body use="literal"/> </input>
   <output> <soap:body use="literal"/> </output>
  </operation>

</binding>

<service name="SHPService">

  <port name="SHPSPort" binding="tns:SHPSBinding">

    <!-- Modify with the REAL SOAP endpoint -->

    <soap:address location="http://example.com/shps"/>

  </port>
```

```
  </service>

</definitions>
```

# References

## Normative

[LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (29 November, 2006). *http://www.projectliberty.org/specs*

[LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification," Version 1.1, Liberty Alliance Project (29 September, 2005). *http://www.projectliberty.org/specs*

[LibertyDP] Cahill, Conor P., eds. "Liberty ID-WSF Design Patterns Specification," Version 1.0, Liberty Alliance Project (14 December 2007). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14 December, 2004). *http://www.projectliberty.org/specs*

[LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. " Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification ," v2.0-errata-1.0-01, 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*

[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*

[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*

[RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2251.txt*

[RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A. Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2252.txt*

[RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search Results," RFC 2891, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2891.txt*

[SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf*

[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*

[SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler, Eve, eds. (15 March, 2005). "Profiles 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-profiles-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/*

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

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004). "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/2004/REC-xml-20040204*

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

[XPATH] Clark , J., DeRose , S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0 ," Recommendation, W3C *http://www.w3.org/TR/xpath [August 2003].*

# Informative

[LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty Alliance Project (13 January, 2005). *http://www.projectliberty.org/specs*

[LibertyIDPPGuide] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Implementation Guidelines," Version 1.1, Liberty Alliance Project (29 September, 2005). *http://www.projectliberty.org/specs*

[LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyACT] Cahill, Conor P., eds. "Liberty Advanced Client Technologies," Version 1.0, Liberty Alliance Project (14 December 2007). *http://www.projectliberty.org/specs*