



Liberty ID-WSF Service Hosting and Proxying Service Specification

Version: 1.0-02

Editors:

Conor Cahill, Intel Corporation

Contributors:

Shelagh Callahan, Intel Corporation

Darryl Champagne, IEEE-ISTO

Jane Dashevsky, Intel Corporation

Hubert Le Van Gong, Sun

Andrew Lindsay-Stewart, Vodafone

Rob Lockhart, IEEE-ISTO

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: draft-liberty-idwsf-shps-v1.0-02.pdf

1 **Notice**

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

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

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

32 Liberty Alliance Project
33 Licensing Administrator
34 c/o IEEE-ISTO
35 445 Hoes Lane
36 Piscataway, NJ 08855-1331, USA
37 info@projectliberty.org

Contents

38		
39	1. Introduction	5
40	1.1. Notation and Conventions	5
41	1.1.1. XML Namespaces	5
42	2. Overview	6
43	3. Service Hosting or Proxying Service Definition	7
44	3.1. Service URIs	7
45	3.2. Data Definitions	7
46	3.2.1. Status Codes	7
47	3.2.2. Request and Response Abstract Types	8
48	3.2.3. Service Mode	9
49	3.2.4. Service Handle	9
50	3.2.5. InvocationContext	10
51	3.2.6. Callback EPR	10
52	3.2.7. Service Descriptor	11
53	3.3. Operation: <i>Query</i>	13
54	3.3.1. wsa:Action values for Query Messages	13
55	3.3.2. Query Message	13
56	3.3.3. QueryResponse Message	14
57	3.3.4. Query Processing Rules	14
58	3.4. Operation: <i>QueryRegistered</i>	14
59	3.4.1. wsa:Action values for QueryRegistered Messages	15
60	3.4.2. QueryRegistered Message	15
61	3.4.3. QueryRegisteredResponse Message	15
62	3.4.4. QueryRegistered Processing Rules	16
63	3.5. Operation: <i>Register</i>	16
64	3.5.1. wsa:Action values for Register Messages	17
65	3.5.2. Register Message	17
66	3.5.3. RegisterResponse Message	18
67	3.5.4. Register Processing Rules	19
68	3.6. Operation: <i>Update</i>	19
69	3.6.1. wsa:Action values for Update Messages	19
70	3.6.2. Update Message	20
71	3.6.3. UpdateResponse Message	21
72	3.6.4. Update Processing Rules	22
73	3.7. Operation: <i>Delete</i>	22
74	3.7.1. wsa:Action values for Delete Messages	22
75	3.7.2. Delete Message	23
76	3.7.3. DeleteResponse Message	23
77	3.7.4. Delete Processing Rules	24
78	3.8. Operation: <i>Invoke</i>	24
79	3.8.1. wsa:Action values for Invoke Messages	24
80	3.8.2. Invoke Message	25
81	3.8.3. InvokeResponse Message	26
82	3.8.4. Invoke Processing Rules	27
83	3.9. Operation: <i>GetStatus</i>	28
84	3.9.1. wsa:Action values for GetStatus Messages	28
85	3.9.2. GetStatus Message	28
86	3.9.3. GetStatusResponse Message	29
87	3.9.4. GetStatus Processing Rules	30
88	3.10. Operation: <i>SetStatus</i>	31
89	3.10.1. wsa:Action values for SetStatus Messages	31
90	3.10.2. SetStatus Message	31

91	3.10.3. SetStatusResponse Message	32
92	3.10.4. SetStatus Processing Rules	33
93	3.11. Operation: <i>Poll</i>	33
94	3.11.1. wsa:Action values for Poll Messages	33
95	3.11.2. Poll Message	33
96	3.11.3. PollResponse Message	34
97	3.11.4. Poll Processing Rules	36
98	3.12. Operation: <i>ProxyInvoke</i>	36
99	3.12.1. wsa:Action values for ProxyInvoke Messages	36
100	3.12.2. ProxyInvoke Message	36
101	3.12.3. ProxyInvokeResponse Message	38
102	3.12.4. ProxyInvoke Processing Rules	40
103	4. Security and Privacy Considerations	42
104	5. Service Hosting/Proxying Service Schema	43
105	6. Service Hosting/Proxying Service WSDL	51
106	References	55

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:2006-12

- 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

136

137 *FIXME(cpc) - Discuss the AC exposing its own services (where the AC is the primary WSP for this service instance*
138 *for the principal). Discuss using the SHPS for improved behavior for network based WSC requests when TM isn't*
139 *available and/or has connectivity changes likely.*

140 *FIXME(cpc) - Discuss how the CSI negotiates with the SHPS for service exposure including: a) Getting SHPS to*
141 *register as provider for this user, b) getting cached data to SHPS, c) updating cached data, d) funneling calls to the*
142 *CSI. Some services will require SHPS to save data and later provide to CSI upon reconnect.*

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 6: 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 5: Service Hosting/Proxying Service Schema](#) .

3.1. Service URIs

Table 1. SHPS URIs

Use	URI
Service Type	<i>urn:liberty:shps:2006-12</i>
Query wsa:Action	<i>urn:liberty:shps:2006-12:Query</i>
QueryResponse wsa:Action	<i>urn:liberty:shps:2006-12:QueryResponse</i>
QueryRegistered wsa:Action	<i>urn:liberty:shps:2006-12:QueryRegistered</i>
QueryRegisteredResponse wsa:Action	<i>urn:liberty:shps:2006-12:QueryRegisteredResponse</i>
Register wsa:Action	<i>urn:liberty:shps:2006-12:Register</i>
RegisterResponse wsa:Action	<i>urn:liberty:shps:2006-12:RegisterResponse</i>
Update wsa:Action	<i>urn:liberty:shps:2006-12:Update</i>
UpdateResponse wsa:Action	<i>urn:liberty:shps:2006-12:UpdateResponse</i>
Delete wsa:Action	<i>urn:liberty:shps:2006-12>Delete</i>
DeleteResponse wsa:Action	<i>urn:liberty:shps:2006-12>DeleteResponse</i>
Invoke wsa:Action	<i>urn:liberty:shps:2006-12:Invoke</i>
InvokeResponse wsa:Action	<i>urn:liberty:shps:2006-12:InvokeResponse</i>
GetStatus wsa:Action	<i>urn:liberty:shps:2006-12:GetStatus</i>
GetStatusResponse wsa:Action	<i>urn:liberty:shps:2006-12:GetStatusResponse</i>
SetStatus wsa:Action	<i>urn:liberty:shps:2006-12:SetStatus</i>
SetStatusResponse wsa:Action	<i>urn:liberty:shps:2006-12:SetStatusResponse</i>
Poll wsa:Action	<i>urn:liberty:shps:2006-12:Poll</i>
PollResponse wsa:Action	<i>urn:liberty:shps:2006-12:PollResponse</i>
ProxyInvoke wsa:Action	<i>urn:liberty:shps:2006-12:ProxyInvoke</i>
ProxyInvokeResponse wsa:Action	<i>urn:liberty:shps:2006-12:ProxyInvokeResponse</i>

3.2. Data Definitions

3.2.1. Status Codes

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

- 159 • Failed
- 160 • Partial
- 161 • OK
- 162 • NeedServiceData
- 163 • NoResults
- 164 • NoSuchService

165 These strings are expected to appear in the "code" attribute of `<lu:Status>` elements used in SOAP-bound SHPS
166 protocol messages [[LibertySOAPBinding](#)]. Specific uses for the status codes are defined in the processing rules
167 for individual messages. The contents of the comment attribute are not defined by this specification, but it may be
168 used for additional descriptive text intended for human consumption (for example, to carry information that will aid
169 debugging).

170 3.2.2. Request and Response Abstract Types

171 3.2.2.1. Complex Type RequestAbstractType

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

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

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

```
177 <!-- RequestAbstractType - common request message structure -->  
178  
179 <xs:complexType name="RequestAbstractType" abstract="true">  
180   <xs:anyAttribute namespace="##other" processContents="lax"/>  
181 </xs:complexType>  
182
```

183 3.2.2.2. Complex Type ResponseAbstractType

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

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

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

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

```
191 <!-- ResponseAbstractType - common message response structure -->
192
193 <xs:complexType name="ResponseAbstractType" abstract="true">
194   <xs:sequence>
195     <xs:element ref="lu:Status"/>
196   </xs:sequence>
197   <xs:anyAttribute namespace="##other" processContents="lax"/>
198 </xs:complexType>
199
```

200 3.2.3. Service Mode

201 A service can be provided by a SHPS in either of two modes:

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

205 This mode is identified by the URN *urn:liberty:shps:2006-12:svcmode:hosted*

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

208 This mode is identified by the URN *urn:liberty:shps:2006-12:svcmode:proxied*

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

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

213 3.2.4. Service Handle

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

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

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

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

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

```
227 <!-- ServiceHandle - a reference to a hosted/proxied service instance -->
228
229 <xs:element name="ServiceHandle" type="xs:anyURI"/>
230
```

231 **Figure 1.** `<shps:ServiceHandle>` — Schema Fragment

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

```
233 <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
234
```

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

236 3.2.5. InvocationContext

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

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

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

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

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

```
249 <!-- InvocationContext - how the proxied service instance was invoked -->
250
251 <xs:element name="InvocationContext" type="InvocationContextType" />
252
253 <xs:complexType name="InvocationContextType">
254   <xs:sequence>
255     <xs:element ref="InvokingProvider" />
256     <xs:element ref="InvokingPrincipal" minOccurs="0" />
257     <xs:element ref="disco:SecurityMechID" />
258   </xs:sequence>
259   <xs:anyAttribute namespace="##other" processContents="lax"/>
260 </xs:complexType>
261
262 <xs:element name="InvokingProvider" type="xs:string" />
263 <xs:element name="InvokingPrincipal" type="saml2:NameIDType" />
264
```

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

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

```
267 <shps:InvocationContext>
268   <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
269   <shps:InvokingPrincipal
270     Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
271     NameQualifier="http://idps-r-us.com" >
272     uuid:23923-023843-230932-230923
273   </shps:InvokingPrincipal>
274   <disco:SecurityMechID>urn:liberty:ds:2006-08:TLS:SAMLV2</disco:SecurityMechID>
275 </shps:InvocationContext>
276
```

277

Example 2. Example <shps:InvocationContext>

278

3.2.6. Callback EPR

279

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]).

280

281

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:

282

283

- 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*

284

285

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

286

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

287

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

288

289

290

Figure 3. <shps:CallbackEPR> — Schema Fragment

291

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

292

```
<shps:CallbackEPR>
```

293

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

294

```
</shps:CallbackEPR>
```

295

296

Example 3. Example anonymous <shps:CallbackEPR>

297

3.2.7. Service Descriptor

298

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:

299

300

- The <wsa:Address> element MUST be *http://www.w3.org/2005/08/addressing/anonymous*

301

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

302

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

303

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

304

305

306

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.

307

308

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

309

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

310

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

311

312

313

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.

314

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

```

316     <!-- ServiceHandle - a reference to a hosted/proxied service instance -->
317
318     <xs:element name="ServiceHandle" type="xs:anyURI" />
319
    
```

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

- The `<ds:Abstract>` and `<ds: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 timeframe, for query responses, it’s the SHPS desired timeframe).

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.

```

347 <wsa:EndpointReference>
348   <wsa:Address>
349     http://www.w3.org/2005/08/addressing/anonymous
350   </wsa:Address>
351
352   <wsa:Metadata>
353     <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:proxied</shps:ServiceMode>
354     <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:hosted</shps:ServiceMode>
355     <ds:ServiceType>urn:liberty:ps:2006-08</ds:ServiceType>
356     <ds:ServiceType>urn:liberty:ps:2007-11</ds:ServiceType>
357     <ds:Framework version="2.0" />
358     <ds:SecurityContext>
359       <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</ds:SecurityMechID>
360       <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</ds:SecurityMechID>
361     </ds:SecurityContext>
362   </wsa:Metadata>
363 </wsa:EndpointReference>
364
    
```

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

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

369 **3.3. Operation: Query**

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

372 **3.3.1. `wsa:Action` values for Query Messages**

373 `<Query>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2006-12:Query."

374 `<QueryResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of
375 "urn:liberty:shps:2006-12:QueryResponse."

376 **3.3.2. Query Message**

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

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

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

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

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

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

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

```
391 <!-- Query - query for ability to host or proxy services -->  
392  
393 <xs:element name="Query" type="disco:QueryType"/>  
394
```

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

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

```

399 <shps:Query>
400   <disco:RequestedService>
401     <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
402     <disco:SecurityMechID>urn:liberty:sm:2006-08:TLS:SAMLV2</disco:SecurityMechID>
403     <disco:Framework version="2.0" />
404     <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:proxied</shps:ServiceMode>
405   </disco:RequestedService>
406 </shps:Query>
407

```

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

409 3.3.3. QueryResponse Message

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

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

```

415 <!--QueryResponse - response for the Query request -->
416
417 <xs:element name="QueryResponse" type="disco:QueryResponseType"/>
418

```

419 **Figure 5. <shps:QueryResponse> — Schema Fragment**

```

420 <shps:QueryResponse>
421   <lu:Status code="OK" />
422   <wsa:EndpointReference>
423     <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
424     <wsa:Metadata>
425       <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:hosted</shps:ServiceMode>
426       <ds:ServiceType>urn:liberty:ps:2006-08</ds:ServiceType>
427       <ds:Framework version="2.0" />
428       <ds:SecurityContext>
429         <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</ds:SecurityMechID>
430       </ds:SecurityContext>
431     </wsa:Metadata>
432   </wsa:EndpointReference>
433 </shps:QueryResponse>
434

```

435 **Example 6. Example <shps:QueryResponse> Message**

436 3.3.4. Query Processing Rules

- 437 • The SHPS SHOULD return Service Descriptions for each service that it is willing to host or proxy which meets
438 the conditions in the request.
- 439 • The SHPS MAY use any factor at its disposal in deciding whether it is willing to host or proxy a service including
440 the identity of the user and/or the Advanced Client itself.
- 441 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code
442 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:2006-12:QueryRegistered."

<QueryRegisteredResponse> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:shps:2006-12: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.

```

461 <!-- QueryRegistered - query for the registered service instances -->
462
463 <xs:element name="QueryRegistered" type="QueryRegisteredType" />
464
465 <xs:complexType name="QueryRegisteredType">
466   <xs:complexContent>
467     <xs:extension base="RequestAbstractType">
468       <xs:sequence>
469         <xs:element ref="ServiceHandle" minOccurs="0" maxOccurs="unbounded" />
470       </xs:sequence>
471     </xs:extension>
472   </xs:complexContent>
473 </xs:complexType>
474

```

Figure 6. <shps:QueryRegistered> — Schema Fragment

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

```

478 <shps:QueryRegistered>
479   <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
480   <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
481 </shps:QueryRegistered>
482

```

Example 7. Example <shps:QueryRegistered> Message

3.4.3. *QueryRegisteredResponse* Message

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

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

```
490 <!-- QueryRegisteredResponse - response for QueryRegistered request -->
491
492 <xs:element name="QueryRegisteredResponse" type="disco:QueryResponseType"/>
493
```

494 **Figure 7. <shps:QueryRegResponse> — Schema Fragment**

```
495 <shps:QueryResponse>
496 <lu:Status code="OK" />
497 <wsa:EndpointReference>
498 <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
499 <wsa:Metadata>
500 <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:hosted</shps:ServiceMode>
501 <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
502 <shps:ServiceStatus>urn:liberty:shps:2006-12:status:enabled</shps:ServiceStatus>
503 <ds:ServiceType>urn:liberty:ps:2006-08</ds:ServiceType>
504 <ds:Framework version="2.0" />
505 <ds:SecurityContext>
506 <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</ds:SecurityMechID>
507 </ds:SecurityContext>
508 </wsa:Metadata>
509 </wsa:EndpointReference>
510 </shps:QueryResponse>
511
```

512 **Example 8. Example <shps:QueryRegisteredResponse> Message**

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

515 3.4.4. QueryRegistered Processing Rules

- 516 • The SHPS MUST only consider service instances registered by this Advanced Client in responding to the request.
 517 Service instances registered by other Advanced Clients MUST NOT be visible to this Advanced Client. There
 518 MUST be no difference in the response to a nonexistent Service Handle and a Service Handle that refers to a
 519 service instance registered by a different Advanced Client.
- 520 • The SHPS SHOULD return Service Descriptions for each registered service instance matching the specified
 521 Service Handle(s). If no Service Handles are specified, Service Descriptors for all of the registered service
 522 instances should be returned.
- 523 • If a specified Service Handle is not currently registered the SHPS SHOULD ignore it and only return the successful
 524 matches.
- 525 • If there were no successful matches of Service Handles, the top-level status code MUST be *Failed* and NO ID-
 526 WSF EPRs are to be returned. In such cases, if the SHPS is providing second level status codes, the second-level
 527 error code MUST be *NoResults*.
- 528 • If there are successful matches, the top-level status code MUST be "OK" and the successfully matching Service
 529 Descriptors are returned.

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

532 **3.5. Operation: *Register***

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

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

538 **3.5.1. wsa:Action values for Register Messages**

539 <Register> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:shps:2006-12:Register."

540 <RegisterResponse> messages MUST include a <wsa:Action> SOAP header with the value of
541 "urn:liberty:shps:2006-12:RegisterResponse."

542 **3.5.2. Register Message**

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

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

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

```
547 <!-- Register - request for a new hosted or proxied service instance -->
548 <xs:element name="Register" type="RegisterType"/>
549 <xs:complexType name="RegisterType">
550   <xs:complexContent>
551     <xs:extension base="RequestAbstractType">
552       <xs:sequence>
553         <xs:element ref="wsa:EndpointReference" maxOccurs="unbounded" />
554       </xs:sequence>
555     </xs:extension>
556   </xs:complexContent>
557 </xs:complexType>
```

561 **Figure 8. <shps:Register> — Schema Fragment**

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

```

565 <shps:Register>
566   <wsa:EndpointReference>
567     <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
568     <wsa:Metadata>
569       <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:hosted</shps:ServiceMode>
570       <ds:ServiceType>urn:liberty:ps:2006-08</ds:ServiceType>
571       <ds:Framework version="2.0" />
572       <ds:SecurityContext>
573         <ds:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</ds:SecurityMechID>
574       </ds:SecurityContext>
575     </wsa:Metadata>
576   </wsa:EndpointReference>
577 </shps:Register>
578

```

579 **Example 9. Example <shps:Register> Message**

580 3.5.3. RegisterResponse Message

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

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

583 <ServiceHandle> **[Optional]** Zero or more service handles, one for each service that the SHPS has agreed to
584 expose on behalf of the Advanced Client.

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

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

```

590 <!-- RegisterResponse - response to the Register request -->
591 <xs:element name="RegisterResponse" type="RegisterResponseType"/>
592
593 <xs:complexType name="RegisterResponseType">
594   <xs:complexContent>
595     <xs:extension base="ResponseAbstractType">
596       <xs:sequence>
597         <xs:element ref="RegisterResponseItem" maxOccurs="unbounded" />
598       </xs:sequence>
599     </xs:extension>
600   </xs:complexContent>
601 </xs:complexType>
602
603 <xs:element name="RegisterResponseItem" type="RegisterResponseItemType" />
604
605 <xs:complexType name="RegisterResponseItemType">
606   <xs:sequence>
607     <xs:element ref="ServiceHandle" />
608   </xs:sequence>
609   <xs:attribute name="ref" type="xs:string" use="required" />
610 </xs:complexType>
611
612

```

613 **Figure 9. <shps:RegisterResponse> — Schema Fragment**

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

```
616 <shps:RegisterResponse>
617   <lu:Status code="OK" />
618   <shps:RegisterResponseItem ref="1">
619     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
620   </shps:RegisterResponseItem>
621   <shps:RegisterResponseItem ref="2">
622     <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238 927</shps:ServiceHandle>
623   </shps:RegisterResponseItem>
624 </shps:RegisterResponse>
625
```

626 **Example 10. Example <shps:RegisterResponse> Message**

627 3.5.4. Register Processing Rules

- 628 • The <shps:Register> request **MUST** include an `itemID` attribute on each <wsa:EndpointReference>
629 element in the request.
630 The value of the `itemID` attribute **MUST** be placed into the `ref` attribute in the <shps:RegisterResponseItem>
631 element associated with the <wsa:EndpointReference> element in the request.
- 632 • The order of the elements in the response **MAY** be different than the order of the associated elements in the request.
633 The `ref` attribute **MUST** be used to correlate the response item to its associated request element.
- 634 • The Advanced Client **MAY** alter a Service Description to have more restrictive settings than those returned from the
635 SHPS in a <shps:QueryResponse>. For example, the Advanced Client may select a single security mechanism
636 for its service instance at the SHPS, even though the SHPS is capable of supporting several different security
637 mechanisms.
638 Any such alterations made **SHOULD NOT** include features or capabilities that the SHPSs has not expressed a
639 willingness to expose.
- 640 • The SHPS **MAY** refuse to accept the registration of any Service Descriptor for which it is unable or unwilling to
641 expose on behalf of the Advanced Client. In such cases the SHPS **MUST** return a failure.
- 642 • The Advanced Client **MAY** initiate a <shps:Register> request using a Service Descriptor that it created
643 (rather than one obtained from the SHP using the <shps:Query> interface). Of course, the SHPS **MAY** refuse
644 such operations if it does not support exposing the requested service or any of the specific features requested.
- 645 • The SHPS **MAY** use any factor at its disposal in deciding whether or not to accept the registration (it **MAY**
646 even deny a registration for a Service Descriptor that the SHPS previously returned to the Advanced Client in a
647 <shps:QueryResponse> if the conditions for such willingness have changed).
- 648 • This operation **MUST** be atomic and if successful, all portions of the request **MUST** have succeeded. If any
649 portion of the request fails, the entire request must fail. This requirement is mostly applicable in the case where
650 the request includes multiple Service Descriptors.
- 651 • If request processing succeeded, the top-level status code **MUST** be "OK." Otherwise, the top-level status code
652 **MUST** be "Failed"
- 653 • If the top-level status code is "Failed," the response **MAY** also contain *Forbidden* as a second-level status code.
654 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will
655 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:2006-12:Update."

`<UpdateResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2006-12: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 10. `<shps:Update>` — Schema Fragment

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

```

701 <shps:Update>
702   <shps:UpdateItem itemID="1">
703     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
704     <wsa:EndpointReference>
705       <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
706       <wsa:Metadata>
707         <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:hosted</shps:ServiceMode>
708         <ds:ServiceType>urn:liberty:ps:2006-08</ds:ServiceType>
709         <ds:Framework version="2.0" />
710         <ds:SecurityContext>
711           <ds:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</ds:SecurityMechID>
712           <ds:SecurityMechID>urn:liberty:security:2006-08:TLS:Bearer</ds:SecurityMechID>
713         </ds:SecurityContext>
714       </wsa:Metadata>
715     </wsa:EndpointReference>
716   </shps:UpdateItem>
717   <shps:UpdateItem itemID="2">
718     <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
719     <wsa:EndpointReference>
720       <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
721       <wsa:Metadata>
722         <shps:ServiceMode>urn:liberty:shps:2006-12:svcmode:hosted</shps:ServiceMode>
723         <ds:ServiceType>urn:liberty:pp:2003-08</ds:ServiceType>
724         <ds:Framework version="2.0" />
725         <ds:SecurityContext>
726           <ds:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</ds:SecurityMechID>
727         </ds:SecurityContext>
728       </wsa:Metadata>
729     </wsa:EndpointReference>
730   </shps:UpdateItem>
731 </shps:Update>
732

```

Example 11. 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`.

```

739 <!-- UpdateResponse - the response to the Update request -->
740
741 <xs:element name="UpdateResponse" type="UpdateResponseType"/>
742
743 <xs:complexType name="UpdateResponseType">
744   <xs:complexContent>
745     <xs:extension base="ResponseAbstractType" />
746   </xs:complexContent>
747 </xs:complexType>
748

```

Figure 11. <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.

```
752 <shps:UpdateResponse>
753   <lu:Status code="Partial">
754     <lu:Status code="NoSuchService" ref="2" />
755   </lu:Status>
756 </shps:UpdateResponse>
757
```

758 **Example 12. Example `<shps:UpdateResponse>` Message**

759 3.6.4. Update Processing Rules

- 760 • If for any reason an update is not accepted, the existing service instance at the SHPS MUST NOT be affected.
- 761 • Service instances created by one Advanced Client MUST NOT be visible to other Advanced Clients. If an
762 Advanced Client presents a Service Handle for a service instance that the Advanced Client did not create, the
763 SHPS MUST behave as if that service instance did not exist and accordingly any attempt to update such a service
764 should fail with an optional error code of *"NotFound."*
- 765 • The SHPS SHOULD refuse to make changes if they indicate features that are not supported by the SHPS instance
766 (such as a security mechanism that the SHPS does not support). In such cases the SHPS MAY indicate this failure
767 case by setting the second-level status code to: *FeatureNotSupported.*
- 768 • The SHPS MAY refuse to accept the update of any service instance for which it is unable or unwilling to expose
769 on behalf of the Advanced Client. In such cases the SHPS MUST return a failure.
- 770 • The Advanced Client MAY initiate a `<shps:Update>` request using a Service Descriptor that it created (rather
771 than one obtained from the SHPS using the `<shps:Query>` interface). Of course, the SHPS MAY refuse such
772 operations if it does not support exposing the requested service or any of the specific features requested.
- 773 • The SHPS MAY use any factor at its disposal in deciding whether or not to accept the update (it MAY even
774 deny an update with a Service Descriptor that the SHPS previously returned to the Advanced Client in a
775 `<shps:QueryResponse>` if the conditions for such willingness has changed).
- 776 • Each `<shps:UpdateItem>` is processed independently and may independently succeed or fail on its own
777 merits.
- 778 • If all `<shps:UpdateItem>` requests are successfully processed, the top-level status code MUST be *"OK."* If
779 all of the items failed, the top-level status code MUST be *"Failed."* Otherwise, if the results were mixed, the top-
780 level status MUST be *"Partial"* and the second level status MUST be included for items for which the processing
781 was not successful indicating so and including the `ref` attribute containing the `itemID` value for the item. These
782 second-level status codes MAY simply be *"Failed,"* or they may indicate with more detail the reason for the failure.
- 783 • If the top-level status code is *"Failed,"* the response MAY also contain other status codes (such as *Forbidden*) as
784 a second-level status code. The SHPS instance may not wish to reveal the reason for failure, in which case no
785 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:2006-12:Delete."

`<DeleteResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2006-12: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 12. `<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:23023-023802-2032023-0238023</shps:ServiceHandle>
  <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
</shps:Delete>
```

Example 13. 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.

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

```
828 <!-- DeleteResponse - response to a delete request -->
829
830 <xs:element name="DeleteResponse" type="DeleteResponseType"/>
831
832 <xs:complexType name="DeleteResponseType">
833   <xs:complexContent>
834     <xs:extension base="ResponseAbstractType" />
835   </xs:complexContent>
836 </xs:complexType>
837
```

838 **Figure 13. <shps:DeleteResponse> — Schema Fragment**

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

```
841 <shps:DeleteResponse>
842   <lu:Status code="Partial">
843     <lu:Status code="NoSuchService"
844       ref="uuid:97326-A726F2-9FE326A-8C34ED3" />
845   </lu:Status>
846 </shps:DeleteResponse>
847
```

848 **Example 14. Example <shps:DeleteResponse> Message**

849 3.7.4. Delete Processing Rules

- 850 • If for any reason an delete is not accepted, the existing service instance at the SHPS **MUST NOT** be affected.
- 851 • Service instances created by one Advanced Client **MUST NOT** be visible to other Advanced Clients. If an
 852 Advanced Client presents a Service Handle for a service instance that the Advanced Client did not create, the
 853 SHPS **MUST** behave as if that service instance did not exist and accordingly any attempt to delete such a service
 854 should fail with an optional error code of *"NotFound."*
- 855 • Each `<shps:ServiceHandle>` is processed independently and may independently succeed or fail on its own
 856 merits.
- 857 • If all of the requested service handles are deleted, the top-level status code **MUST** be *"OK."* If all of the deletes
 858 failed, the top-level status code **MUST** be *"Failed."* Otherwise, if the results were mixed, the top-level status **MUST**
 859 be *"Partial"* and the second level status **MUST** be included for items for which the processing was not successful
 860 indicating so and including the `ref` attribute containing the `ServiceHandle` value for the delete(s) that failed.
 861 These second-level status codes **MAY** simply be *"Failed,"* or they may indicate with more detail the reason for the
 862 failure.
- 863 • If the top-level status code is *"Failed,"* the response **MAY** also contain other status codes (such as *Forbidden*) as a
 864 second-level status code. The SHPS may not wish to reveal the reason for failure, in which case no second-level
 865 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:2006-12:Invoke."

`<InvokeResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2006-12: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 requestor (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`.

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

```

896 <!-- Invoke - invoke service level interface on hosted service instance -->
897
898 <xs:element name="Invoke" type="InvokeType" />
899
900 <xs:complexType name="InvokeType">
901   <xs:complexContent>
902     <xs:extension base="RequestAbstractType">
903       <xs:sequence>
904         <xs:element ref="InvokeItem" maxOccurs="unbounded" />
905       </xs:sequence>
906     </xs:extension>
907   </xs:complexContent>
908 </xs:complexType>
909
910 <!-- InvokeItem - container for each service level request -->
911
912 <xs:element name="InvokeItem" type="InvokeItemType" />
913
914 <xs:complexType name="InvokeItemType">
915   <xs:sequence>
916     <xs:element ref="ServiceHandle" />
917     <xs:any namespace="##other"
918       processContents="lax"
919       minOccurs="0"
920       maxOccurs="unbounded" />
921   </xs:sequence>
922   <xs:attribute name="itemID" type="xs:string" use="required" />
923   <xs:anyAttribute namespace="##other" processContents="lax" />
924 </xs:complexType>
925

```

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

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

```

930 <shps: Invoke>
931   <shps: InvokeItem itemID="1">
932     <shps: ServiceHandle>uuid:23023-023802-2032023-0238023</shps: ServiceHandle>
933     <pp:Modify>
934       <pp:ModifyItem>
935         ... modification data goes here ...
936       </pp:ModifyItem>
937     </pp:Modify>
938   </shps: InvokeItem>
939 </shps: Invoke>
940

```

941 **Example 15.** Example `<shps: Invoke>` Message

942 3.8.3. InvokeResponse Message

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

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

945 `<shps: InvokeResponseItem>`: **[Optional]** The container element for the response data for an
 946 `<shps: InvokeItem>` which contains the following attributes/elements:

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

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

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

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

```

959 <!-- InvokeResponse - response to an Invoke request -->
960 <xs:element name="InvokeResponse" type="InvokeResponseType" />
961
962 <xs:complexType name="InvokeResponseType">
963   <xs:complexContent>
964     <xs:extension base="ResponseAbstractType">
965       <xs:sequence>
966         <xs:element ref="InvokeResponseItem" maxOccurs="unbounded" />
967       </xs:sequence>
968     </xs:extension>
969   </xs:complexContent>
970 </xs:complexType>
971
972 <!-- InvokeResponseItem - container for the result of each InvokeItem -->
973 <xs:element name="InvokeResponseItem" type="InvokeResponseItemType" />
974
975 <xs:complexType name="InvokeResponseItemType">
976   <xs:sequence>
977     <xs:any namespace="##other"
978       processContents="lax"
979       minOccurs="0"
980       maxOccurs="unbounded" />
981   </xs:sequence>
982   <xs:attribute name="ref" type="xs:string" use="required" />
983 </xs:complexType>
984
985
986
  
```

987 **Figure 15. <shps:InvokeResponse> — Schema Fragment**

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

```

990 <shps:InvokeResponse>
991   <lu:Status code="OK" />
992   <shps:InvokeResponseItem ref="1">
993     <pp:ModifyResponse>
994       ... modification response data goes here ...
995     </pp:ModifyResponse>
996   </shps:InvokeResponseItem>
997 </shps:InvokeResponse>
998
  
```

999 **Example 16. Example <shps:InvokeResponse> Message**

1000 3.8.4. Invoke Processing Rules

- 1001 • The Advanced Client MAY invoke any service interface necessary to populate the SHPS hosted service instance
1002 with sufficient data to properly respond to requests from WSCs.
- 1003 • Each `<shps:InvokeResponse>` element in the request MUST be treated independently.
- 1004 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code
1005 MUST be "Failed"
- 1006 • If the top-level status code is "Failed," the response MAY also contain *Forbidden* as a second-level status code.
1007 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will
1008 appear.

1009 3.9. Operation: *GetStatus*

1010 The *GetStatus* operation is used by the Advanced Client to determine the status of a registered service instance at the
1011 SHPS.

1012 3.9.1. `wsa:Action` values for *GetStatus* Messages

1013 `<GetStatus>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2006-12:GetStatus"

1014 `<GetStatusResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of
1015 "urn:liberty:shps:2006-12:GetStatusResponse."

1016 3.9.2. *GetStatus* Message

1017 The `<GetStatus>` request is called to get the current status of the specified service instance at the SHPS.

1018 The `<shps:GetStatus>` request contains the following attributes and/or elements:

- 1019 • `<shps:ServiceHandle>` **[Optional]** Zero or more Service Handles for the service instances who's status is
1020 requested. If not specified, the status of all service instances registered by the calling Advanced Client should be
1021 returned.
- 1022 • `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
1023 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

1024 The schema for the `<shps:GetStatus>` is shown below.

```
1025 <!-- GetStatus - Get the status of a Service Instance at SHPS -->
1026
1027 <xs:element name="GetStatus" type="GetStatusType" />
1028
1029 <xs:complexType name="GetStatusType">
1030   <xs:complexContent>
1031     <xs:extension base="RequestAbstractType">
1032       <xs:sequence>
1033         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1034       </xs:sequence>
1035     </xs:extension>
1036   </xs:complexContent>
1037 </xs:complexType>
1038
```

1039 **Figure 16. `<shps:GetStatus>` — Schema Fragment**

1040 An example message body containing a `<GetStatus>` message follows.

```
1041 <shps:GetStatus>
1042   <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1043   <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
1044 </shps:GetStatus>
1045
```

1046 **Example 17. Example `<shps:GetStatus>` Message**

1047 **3.9.3. GetStatusResponse Message**

1048 This response to the `<shps:GetStatus>` request contains the following elements and/or attributes:

- 1049 • `<lu:Status>`: **[Required]** - the status of the response. See the processing rules below for more information.
- 1050 • `<shps:GetStatusResponseItem>` : **[Optional]** - zero or more Service Handle/Service Status pairs indicating
1051 the current status of the specified service. This element MAY be absent if there were no service instances
1052 registered at the SHPS that met the conditions of the query or if the call otherwise failed.
- 1053 • `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One
1054 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

```

1055 <!-- GetStatusResponse - response to the GetStatus request -->
1056
1057 <xs:element name="GetStatusResponse" type="GetStatusResponseType" />
1058
1059 <xs:complexType name="GetStatusResponseType">
1060   <xs:complexContent>
1061     <xs:extension base="ResponseAbstractType">
1062       <xs:sequence>
1063         <xs:element ref="GetStatusResponseItem" minOccurs="0"
1064           maxOccurs="unbounded" />
1065       </xs:sequence>
1066     </xs:extension>
1067   </xs:complexContent>
1068 </xs:complexType>
1069
1070 <!-- GetStatusResponseItem - the status of a single service instance -->
1071
1072 <xs:element name="GetStatusResponseItem" type="GetStatusResponseItemType" />
1073
1074 <xs:complexType name="GetStatusResponseItemType">
1075   <xs:sequence>
1076     <xs:element ref="ServiceHandle" />
1077     <xs:element ref="ServiceStatus" />
1078   </xs:sequence>
1079   <xs:anyAttribute namespace="##other" processContents="lax" />
1080 </xs:complexType>
1081
1082

```

Figure 17. <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.

```

1086 <shps:GetStatusResponse>
1087   <lu:Status code="OK" />
1088   <shps:GetStatusResponseItem>
1089     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1090     <shps:ServiceStatus>urn:liberty:shps:2006-12:status:enabled</shps:ServiceStatus>
1091   </shps:GetStatusResponseItem>
1092   <shps:GetStatusResponseItem>
1093     <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
1094     <shps:ServiceStatus>urn:liberty:shps:2006-12:status:enabled</shps:ServiceStatus>
1095   </shps:GetStatusResponseItem>
1096 </shps:GetStatusResponse>
1097

```

Example 18. 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.

1102 • A request for a service instance which is NOT registered for the Advanced Client (whether or not it is reg-
1103 istered for another Advanced Client) is, for the SHPS, considered successful and simply does not result in a
1104 <GetStatusResponseItem> in the results of the operation. For example, if the <GetStatus> included a
1105 single <ServiceHandle> referring to a service instance not currently registered at the SHPS, the response would
1106 be:

```
1107 <shps:GetStatusResponse>  
1108 <lu:Status code="OK" />  
1109 </shps:GetStatusResponse>  
1110
```

1111 • If there are any matching results (even if there are some other non-matching elements), the request is successful
1112 and the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.

1113 • If the top-level status code is "*Failed*," the response MAY also contain *NoResults* as a second-level status code.
1114 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will
1115 appear.

1116 3.10. Operation: *SetStatus*

1117 The *SetStatus* operation is used by the Advanced Client to enable or disable a registered service instance at the SHPS.

1118 This interface is typically used by the Advanced Client after it has completed the initialization of the hosted/proxied
1119 service to make the service visible to WSCs.

1120 The primary effect of this call is that the SHPS registers and associates (or disassociates) the hosted or proxied service
1121 instance in the principal's DS resource. This makes the service instance visible (or invisible) to WSCs.

1122 3.10.1. *wsa:Action* values for *SetStatus* Messages

1123 <SetStatus> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:shps:2006-12:SetStatus"

1124 <SetStatusResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1125 "urn:liberty:shps:2006-12:SetStatusResponse."

1126 3.10.2. *SetStatus* Message

1127 The <SetStatus> request is called to set the current status of the specified service instance to the specified status.

1128 The <shps:SetStatus> request contains the following attributes and/or elements:

1129 • *SetStatusItem*: **[Required]** - one or more set status items each of which contain:

1130 • *ServiceStatus*: **[Required]** - the new status for the specified service instance(s). This MUST be set to one
1131 of the following values:

1132 • *urn:liberty:shps:2006-12:status:enabled*

1133 The service instance is to be made visible in the principal's DS resource such that WSCs can discover and
1134 invoke it.

1135 • *urn:liberty:shps:2006-12:status:disabled*

1136 The service instance is to be removed from the principal's DS resource such that WSCs can no longer
1137 discover and invoke it.

1138 • <shps:ServiceHandle> **[Required]** - one or more Service Handles for the service instances which are to
1139 be enabled or disabled.

1140 • anyAttribute **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
 1141 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

1142 The schema for the `<shps:SetStatus>` is shown below.

```

1143 <!-- SetStatus - set the status of a hosted/proxied service instance -->
1144
1145 <xs:element name="SetStatus" type="SetStatusType"/>
1146
1147 <xs:complexType name="SetStatusType">
1148   <xs:complexContent>
1149     <xs:extension base="RequestAbstractType">
1150       <xs:sequence>
1151         <xs:element ref="SetStatusItem" maxOccurs="unbounded" />
1152       </xs:sequence>
1153     </xs:extension>
1154   </xs:complexContent>
1155 </xs:complexType>
1156
1157 <xs:element name="SetStatusItem" type="SetStatusItemType"/>
1158
1159 <xs:complexType name="SetStatusItemType">
1160   <xs:sequence>
1161     <xs:element ref="ServiceStatus" />
1162     <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1163   </xs:sequence>
1164   <xs:attribute name="itemID" type="xs:string" use="required" />
1165 </xs:complexType>
1166

```

1167 **Figure 18.** `<shps:SetStatus>` — Schema Fragment

1168 An example message body containing a `<SetStatus>` message follows. This request enables two service instances.

```

1169 <shps:SetStatus>
1170   <shps:SetStatusItem itemID="1" >
1171     <shps:ServiceStatus>urn:liberty:shps:2006-12:status:disabled</shps:ServiceStatus>
1172     <shps:ServiceHandle>uuid:53723-123872-5223223-8237823</shps:ServiceHandle>
1173   </shps:SetStatusItem>
1174   <shps:SetStatusItem itemID="2">
1175     <shps:ServiceStatus>urn:liberty:shps:2006-12:status:enabled</shps:ServiceStatus>
1176     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1177     <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
1178   </shps:SetStatusItem>
1179 </shps:SetStatus>
1180

```

1181 **Example 19.** Example `<shps:SetStatus>` Message

1182 3.10.3. SetStatusResponse Message

1183 This response to the `<shps:SetStatus>` request contains the following elements:

1184 `<lu:Status>`: **[Required]** The status of the response. See the processing rules below for more information.

1185 anyAttribute **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
 1186 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

```
1187 <!-- SetStatusResponse - response to the SetStatus request -->
1188
1189 <xs:element name="SetStatusResponse" type="SetStatusResponseType" />
1190
1191 <xs:complexType name="SetStatusResponseType">
1192   <xs:complexContent>
1193     <xs:extension base="ResponseAbstractType" />
1194   </xs:complexContent>
1195 </xs:complexType>
1196
```

Figure 19. `<shps:SetStatusResponse>` — Schema Fragment

1198 An example message body containing a `<SetStatusResponse>` message follows. This is a successful response.

```
1199 <shps:SetStatusResponse>
1200   <lu:Status code="OK" />
1201 </shps:SetStatusResponse>
1202
```

Example 20. Example `<shps:SetStatusResponse>` Message

1204 3.10.4. SetStatus Processing Rules

- 1205 • This operation **MUST** be atomic and if successful, all portions of the request **MUST** have succeeded. If any
1206 portion of the request fails, the entire request must fail. This requirement is mostly applicable in the case where
1207 the request includes multiple Service Descriptors.
- 1208 • Each `<shps:SetStatusItem>` **MUST** be processed independently (even if one fails, the other
1209 `<shps:SetStatusItem>` elements **MUST** be processed.
- 1210 • If request processing succeeded, the top-level status code **MUST** be "OK." Otherwise, the top-level status code
1211 **MUST** be "Failed" If all `<shps:SetStatusItem>` requests are successfully processed, the top-level status
1212 code **MUST** be "OK." If all of the items failed, the top-level status code **MUST** be "Failed." Otherwise, if the
1213 results were mixed, the top-level status **MUST** be "Partial" and the second level status **MUST** be included for
1214 items for which the processing was not successful indicating so and including the `ref` attribute containing the
1215 `itemID` value for the item. These second-level status codes **MAY** simply be "Failed," or they may indicate with
1216 more detail the reason for the failure.
- 1217 • If the top-level status code is "Failed," the response **MAY** also contain *NotFound* as a second-level status code.
1218 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will
1219 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:2006-12:Poll."

`<PollResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2006-12: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 20. `<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:2006-12:ProxyInvoke</wsa:Action>
</shps:Poll>
```

Example 21. 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.

```

1248 <shps:Poll wait="300">
1249   <wsa:Action>urn:liberty:shps:2006-12:ProxyInvoke</wsa:Action>
1250   <dp:Response ref="1">
1251     <shps:ProxyInvokeResponse>
1252       <lu:Status code="OK" />
1253       <shps:ProxyInvokeResponseItem ref="1">
1254         <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1255         <shps:ResponseHeaders>
1256           <sb:UsageDirectives> . . . . . </sb:UsageDirectives>
1257         </shps:ResponseHeaders>
1258         <pp:QueryResponse>
1259           . . . modification response data goes here . . .
1260         </pp:QueryResponse>
1261       </shps:ProxyInvokeResponseItem>
1262     </shps:ProxyInvokeResponse>
1263   </dp:Response>
1264 </shps:Poll>
1265

```

Example 22. 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.

```

1270 <!--PollResponse - response for the Poll request -->
1271
1272 <xs:element name="PollResponse" type="dp:PollResponseType"/>
1273

```

Figure 21. <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).

```

1278 <shps:PollResponse nextPoll="600">
1279   <lu:Status code="OK" />
1280 </shps:PollResponse>
1281

```

Example 23. 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.

```

1285 <shps:PollResponse>
1286   <lu:Status code="OK" />
1287   <dp:Request itemID="1">
1288     <shps:ProxyInvoke>
1289       <shps:ProxyInvokeItem itemID="1">
1290         <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1291         <shps:InvocationContext>
1292           <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
1293           <shps:InvokingPrincipal
1294             Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
1295             NameQualifier="http://idps-r-us.com" >
1296             uuid:23923-023843-230932-230923
1297           </shps:InvokingPrincipal>
1298           <disco:SecurityMechID>urn:liberty:ds:2006-08:TLS:SAMLV2</disco:SecurityMechID>
1299         </shps:InvocationContext>
1300         <shps:RequestHeaders>
1301           <sb:ProcessingContext>
1302             urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1303           </sb:ProcessingContext>
1304         </shps:RequestHeaders>
1305         <pp:Query>
1306           ... query data goes here ...
1307         </pp:Query>
1308       </shps:ProxyInvokeItem>
1309     </shps:ProxyInvoke>
1310   </dp:Request>
1311 </shps:PollResponse>
1312

```

1313 **Example 24. Example <shps:PollResponse> Message**

1314 **3.11.4. Poll Processing Rules**

- 1315 • All of the processing rules defined for the *Poll* design pattern MUST be followed. See [LibertyDP] for further
1316 information.

1317 **3.12. Operation: *ProxyInvoke***

1318 The *ProxyInvoke* operation is NOT exposed by the SHPS service, but rather MUST be exposed by an Advanced Client
1319 that uses the proxying capability of SHPS. This interface is what SHPS calls when it has been invoked in order to get
1320 the data to use in the response.

1321 The Advanced Client must either expose this interface directly via an externally visible endpoint (specified in the
1322 <shps:CallbackEPR> – see Section 3.2.6) or MUST use the <shps:Poll> interface to poll for invocations.

1323 **3.12.1. wsa:Action values for ProxyInvoke Messages**

1324 <ProxyInvoke> messages MUST include a <wsa:Action> SOAP header with the value of
1325 "urn:liberty:shps:2006-12:ProxyInvoke."

1326 <ProxyInvokeResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1327 "urn:liberty:shps:2006-12:ProxyInvokeResponse."

1328 **3.12.2. ProxyInvoke Message**

1329 The <ProxyInvoke> request is called by the SHPS to pass on an invocation received from a WSC so that the
1330 Advanced Client can process the request and send the results to the SHPS who can then respond to the WSC.

1331 The <shps:ProxyInvoke> request contains one or more <shps:ProxyInvokeItem> elements each of which
1332 have the following attributes and/or elements:

- 1333 • `itemID`: **[Required]** - a string that may have any value assigned by the requestor (but **MUST** be different than the
1334 `itemID` assigned to any other `<shps:ProxyInvokeItem>` elements in the same request.
- 1335 The `itemID` is used to enable correlation of the results in the `<shps:ProxyInvokeResponse>` with the
1336 `<shps:ProxyInvokeItem>` that lead to those results.
- 1337 • `<shps:ServiceHandle>` **[Required]** - the Service Handle for the service instance which this
1338 `<shps:ProxyInvokeItem>` applies to. Different Service Handles **MAY** be specified in differ-
1339 ent `<shps:ProxyInvokeItem>` elements in the same request if the Advanced Client used the same
1340 `<shps:CallbackEPR>` for multiple service instances and multiple simultaneous invocations took place at the
1341 SHPS.
- 1342 • `<shps:InvocationContext>` **[Required]** - the invocation context (see [Section 3.2.5](#) of the service instance at
1343 the SHPS (necessary information for the CSI to make appropriate decisions about the results of the operation).
- 1344 • `<shps:RequestHeaders>` **[Optional]** - a container for zero or more SOAP headers from the call received by the
1345 SHPS which the SHPS considers necessary for the CSI to resolve the request. Typical candidate headers include
1346 (but are **not** limited to):
- 1347 • `<sb:ApplicationEPR>`
 - 1348 • `<sb:Consent>`
 - 1349 • `<sb:ProcessingContext>`
 - 1350 • `<sb:Timeout>`
 - 1351 • `<sb:UsageDirectives>`
- 1352 • `<xs:any>` **[Required]** - the unmodified incoming request that the SHPS received from the WSC. For example, if
1353 the Liberty Profile Service is the service that is being proxied by the SHPS, an `<pp:Query>` may be specified
1354 in this location.
- 1355 The individual service definitions (such as the Liberty Personal Profile Service Specification) drive what is
1356 allowable within this location. Essentially, anything that **MAY** be placed into the body of a typical invocation of
1357 the service **MAY** be placed into the `<shps:ProxyInvokeItem>` element.
- 1358 • `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
1359 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

1360 The schema for the `<shps:ProxyInvoke>` is shown below.

```

1361 <!-- ProxyInvoke - proxied invocation of CSI -->
1362
1363 <xs:element name="ProxyInvoke" type="ProxyInvokeType" />
1364
1365 <xs:complexType name="ProxyInvokeType">
1366   <xs:complexContent>
1367     <xs:extension base="RequestAbstractType">
1368       <xs:sequence>
1369         <xs:element ref="ProxyInvokeItem" maxOccurs="unbounded" />
1370       </xs:sequence>
1371     </xs:extension>
1372   </xs:complexContent>
1373 </xs:complexType>
1374
1375
1376 <!-- Declaration of ProxyInvokeItem element -->
1377 <xs:element name="ProxyInvokeItem" type="ProxyInvokeItemType" />
1378
1379 <xs:complexType name="ProxyInvokeItemType">
1380   <xs:sequence>
1381     <xs:element ref="ServiceHandle" />
1382     <xs:element ref="InvocationContext" />
1383     <xs:element ref="RequestHeaders" minOccurs="0" />
1384     <xs:any namespace="##other"
1385       processContents="lax"
1386       minOccurs="0"
1387       maxOccurs="unbounded" />
1388   </xs:sequence>
1389   <xs:attribute name="itemID" type="xs:string" use="required" />
1390   <xs:anyAttribute namespace="##other" processContents="lax" />
1391 </xs:complexType>
1392
1393 <xs:element name="RequestHeaders" />
1394

```

1395 **Figure 22.** `<shps:ProxyInvoke>` — Schema Fragment

1396 An example message body containing a `<ProxyInvoke>` message follows. This is a proxied Liberty Personal Profile
 1397 `<pp:Query>` request on the service instance referenced by the specified `<shps:ServiceHandle>`).

```

1398 <shps:ProxyInvoke>
1399   <shps:ProxyInvokeItem itemID="1">
1400     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1401     <shps:InvocationContext>
1402       <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
1403       <shps:InvokingPrincipal
1404         Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
1405         NameQualifier="http://idps-r-us.com" >
1406         uuid:23923-023843-230932-230923
1407       </shps:InvokingPrincipal>
1408       <disco:SecurityMechID>urn:liberty:ds:2006-08:TLS:SAMLV2</disco:SecurityMechID>
1409     </shps:InvocationContext>
1410     <shps:RequestHeaders>
1411       <sb:ProcessingContext>
1412         urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1413       </sb:ProcessingContext>
1414     </shps:RequestHeaders>
1415     <pp:Query>
1416       ... query data goes here ...
1417     </pp:Query>
1418   </shps:ProxyInvokeItem>
1419 </shps:ProxyInvoke>
1420

```

1421

Example 25. Example <shps:ProxyInvoke> Message

1422

3.12.3. ProxyInvokeResponse Message

1423

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

1424

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

1425

- <shps:ProxyInvokeResponseItem> : **[Optional]** The container element for the response data for an <shps:ProxyInvokeItem> which contains the following attributes/elements:

1426

1427

- 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.

1428

1429

- <shps:ServiceHandle> **[Required]** - the Service Handle for the service instance which this <shps:ProxyInvokeResponseItem> applies to.

1430

1431

- <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):

1432

1433

- <sb:CredentialsContext>

1434

- <sb:UsageDirectives>

1435

- <sb:UserInteraction>

1436

- <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.

1437

1438

1439

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.

1440

1441

1442

- 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.

1443

```

1444 <!-- ProxyInvokeResponse - response to the ProxyInvoke request -->
1445
1446 <xs:element name="ProxyInvokeResponse" type="ProxyInvokeResponseType" />
1447
1448 <xs:complexType name="ProxyInvokeResponseType">
1449   <xs:complexContent>
1450     <xs:extension base="ResponseAbstractType">
1451       <xs:sequence>
1452         <xs:element ref="ProxyInvokeResponseItem" minOccurs="0"
1453           maxOccurs="unbounded" />
1454       </xs:sequence>
1455     </xs:extension>
1456   </xs:complexContent>
1457 </xs:complexType>
1458
1459 <!-- ProxyInvokeResponseItem - container for each service level invocation -->
1460
1461 <xs:element name="ProxyInvokeResponseItem" type="ProxyInvokeResponseItemType" />
1462
1463 <xs:complexType name="ProxyInvokeResponseItemType">
1464   <xs:sequence>
1465     <xs:element ref="ServiceHandle" />
1466     <xs:element ref="ResponseHeaders" minOccurs="0" />
1467     <xs:any namespace="##other"
1468       processContents="lax"
1469       minOccurs="0"
1470       maxOccurs="unbounded" />
1471   </xs:sequence>
1472   <xs:attribute name="ref" type="xs:string" use="required" />
1473 </xs:complexType>
1474
1475 <xs:element name="ResponseHeaders" />
1476

```

Figure 23. `<shps:ProxyInvokeResponse>` — Schema Fragment

1478 An example message body containing a `<ProxyInvokeResponse>` message follows. This is a successful response
1479 for a Liberty Personal Profile query request.

```

1480 <shps:ProxyInvokeResponse>
1481   <lu:Status code="OK" />
1482   <shps:ProxyInvokeResponseItem ref="1">
1483     <shps:ServiceHandle>uuid:23023-023802-2032023-0 238023</shps:ServiceHandle>
1484     <shps:ResponseHeaders>
1485       <sb:UsageDirectives> . . . . . </sb:UsageDirectives>
1486     </shps:ResponseHeaders>
1487     <pp:QueryResponse>
1488       . . . modification response data goes here . . .
1489     </pp:QueryResponse>
1490   </shps:ProxyInvokeResponseItem>
1491 </shps:ProxyInvokeResponse>
1492

```

Example 26. Example `<shps:ProxyInvokeResponse>` Message

1494 3.12.4. ProxyInvoke Processing Rules

- 1495 • The SHPS MUST build the `<shps:InvocationContext>` from the message it received from the WSC.
- 1496 • Each `<shps:ProxyInvokeItem>` MUST be processed independently (even if one fails, the other
1497 `<shps:ProxyInvokeItem>` elements MUST be processed.

- 1498
- 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).
- 1499
- 1500
- 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.
- 1501
- 1502
- 1503
- 1504
- 1505
- 1506
- 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).
- 1507
- 1508
- 1509
- 1510
- 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.
- 1511
- 1512

1513 **4. Security and Privacy Considerations**

1514 This section describes security and privacy related considerations that should be taken into account when implementing
1515 and deploying environments making use of the components specified in this document.

1516 The order of entries in this list has no significance.

1517 1. consideration goes here

5. Service Hosting/Proxying Service Schema

1518

```
1519 <?xml version="1.0" encoding="UTF-8"?>
1520
1521 <xs:schema targetNamespace="urn:liberty:shps:2006-12"
1522
1523     xmlns:lu="urn:liberty:util:2006-08"
1524
1525     xmlns:disco="urn:liberty:disco:2006-08"
1526
1527     xmlns:dp="urn:liberty:dp:2006-12"
1528
1529     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1530
1531     xmlns:wsa="http://www.w3.org/2005/08/addressing"
1532
1533     xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
1534
1535
1536
1537     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
1538
1539     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1540
1541
1542
1543     xmlns="urn:liberty:shps:2006-12"
1544
1545     elementFormDefault="qualified"
1546
1547     attributeFormDefault="unqualified"
1548
1549 >
1550
1551
1552
1553 <xs:import namespace="urn:liberty:util:2006-08"
1554
1555     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1556
1557
1558
1559 <xs:import namespace="urn:liberty:disco:2006-08"
1560
1561     schemaLocation="liberty-idwsf-disco-svc-v2.0.xsd"/>
1562
1563
1564
1565 <xs:import namespace="urn:liberty:dp:2006-12"
1566
1567     schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>
1568
1569
1570
1571 <xs:import namespace="http://www.w3.org/2005/08/addressing"
1572
1573     schemaLocation="http://www.w3.org/2005/08/addressing/ws-addr.xsd" />
1574
1575
1576
1577 <xs:import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
1578
1579     schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd" />
1580
1581
1582 <!-- Data Definitions { -->
1583
```

```
1584 <!-- ServiceHandle - a reference to a hosted/proxied service instance -->
1585
1586 <xs:element name="ServiceHandle" type="xs:anyURI" />
1587
1588
1589 <!-- ServiceMode - the proxied or hosted mode of the service instance -->
1590
1591 <xs:element name="ServiceMode" type="xs:anyURI" />
1592
1593
1594 <!-- ServiceStatus - the enabled/disabled status of the service instance -->
1595
1596 <xs:element name="ServiceStatus" type="xs:anyURI" />
1597
1598
1599 <!-- InvocationContext - how the proxied service instance was invoked -->
1600
1601 <xs:element name="InvocationContext" type="InvocationContextType" />
1602
1603 <xs:complexType name="InvocationContextType">
1604   <xs:sequence>
1605     <xs:element ref="InvokingProvider" />
1606     <xs:element ref="InvokingPrincipal" minOccurs="0" />
1607     <xs:element ref="disco:SecurityMechID" />
1608   </xs:sequence>
1609   <xs:anyAttribute namespace="##other" processContents="lax" />
1610 </xs:complexType>
1611
1612 <xs:element name="InvokingProvider" type="xs:string" />
1613 <xs:element name="InvokingPrincipal" type="saml2:NameIDType" />
1614
1615
1616 <!-- CallbackEPR - where the CSI can receive ProxyInvoke requests -->
1617
1618 <xs:element name="CallbackEPR" type="wsa:EndpointReferenceType" />
1619
1620
1621
1622 <!-- End of Data Definitions } -->
1623
1624 <!-- Interface Definitions { -->
1625
1626
1627
1628 <!-- RequestAbstractType - common request message structure -->
1629
1630 <xs:complexType name="RequestAbstractType" abstract="true">
1631   <xs:anyAttribute namespace="##other" processContents="lax" />
1632 </xs:complexType>
1633
1634
1635 <!-- ResponseAbstractType - common message response structure -->
1636
1637 <xs:complexType name="ResponseAbstractType" abstract="true">
1638   <xs:sequence>
1639     <xs:element ref="lu:Status" />
1640   </xs:sequence>
1641   <xs:anyAttribute namespace="##other" processContents="lax" />
1642 </xs:complexType>
1643
1644
1645 <!-- Delete - delete (remove) a service instance -->
1646
1647 <xs:element name="Delete" type="DeleteType" />
1648
1649 <xs:complexType name="DeleteType">
1650   <xs:complexContent>
```

```

1651     <xs:extension base="RequestAbstractType">
1652       <xs:sequence>
1653         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1654       </xs:sequence>
1655     </xs:extension>
1656   </xs:complexContent>
1657 </xs:complexType>
1658
1659
1660 <!-- DeleteResponse - response to a delete request -->
1661
1662 <xs:element name="DeleteResponse" type="DeleteResponseType"/>
1663
1664 <xs:complexType name="DeleteResponseType">
1665   <xs:complexContent>
1666     <xs:extension base="ResponseAbstractType" />
1667   </xs:complexContent>
1668 </xs:complexType>
1669
1670
1671 <!-- GetStatus - Get the status of a Service Instance at SHPS -->
1672
1673 <xs:element name="GetStatus" type="GetStatusType"/>
1674
1675 <xs:complexType name="GetStatusType">
1676   <xs:complexContent>
1677     <xs:extension base="RequestAbstractType">
1678       <xs:sequence>
1679         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1680       </xs:sequence>
1681     </xs:extension>
1682   </xs:complexContent>
1683 </xs:complexType>
1684
1685
1686 <!-- GetStatusResponse - response to the GetStatus request -->
1687
1688 <xs:element name="GetStatusResponse" type="GetStatusResponseType"/>
1689
1690 <xs:complexType name="GetStatusResponseType">
1691   <xs:complexContent>
1692     <xs:extension base="ResponseAbstractType">
1693       <xs:sequence>
1694         <xs:element ref="GetStatusResponseItem" minOccurs="0"
1695           maxOccurs="unbounded" />
1696       </xs:sequence>
1697     </xs:extension>
1698   </xs:complexContent>
1699 </xs:complexType>
1700
1701 <!-- GetStatusResponseItem - the status of a single service instance -->
1702
1703 <xs:element name="GetStatusResponseItem" type="GetStatusResponseItemType"/>
1704
1705 <xs:complexType name="GetStatusResponseItemType">
1706   <xs:sequence>
1707     <xs:element ref="ServiceHandle" />
1708     <xs:element ref="ServiceStatus" />
1709   </xs:sequence>
1710   <xs:anyAttribute namespace="##other" processContents="lax"/>
1711 </xs:complexType>
1712
1713
1714
1715 <!-- Query - query for ability to host or proxy services -->
1716
1717 <xs:element name="Query" type="disco:QueryType"/>

```

```
1718
1719
1720 <!--QueryResponse - response for the Query request -->
1721
1722 <xs:element name="QueryResponse" type="disco:QueryResponseType"/>
1723
1724
1725 <!-- Invoke - invoke service level interface on hosted service instance -->
1726
1727 <xs:element name="Invoke" type="InvokeType"/>
1728
1729 <xs:complexType name="InvokeType">
1730   <xs:complexContent>
1731     <xs:extension base="RequestAbstractType">
1732       <xs:sequence>
1733         <xs:element ref="InvokeItem" maxOccurs="unbounded" />
1734       </xs:sequence>
1735     </xs:extension>
1736   </xs:complexContent>
1737 </xs:complexType>
1738
1739 <!-- InvokeItem - container for each service level request -->
1740
1741 <xs:element name="InvokeItem" type="InvokeItemType"/>
1742
1743 <xs:complexType name="InvokeItemType">
1744   <xs:sequence>
1745     <xs:element ref="ServiceHandle" />
1746     <xs:any namespace="##other"
1747       processContents="lax"
1748       minOccurs="0"
1749       maxOccurs="unbounded" />
1750   </xs:sequence>
1751   <xs:attribute name="itemID" type="xs:string" use="required" />
1752   <xs:anyAttribute namespace="##other" processContents="lax" />
1753 </xs:complexType>
1754
1755
1756 <!-- InvokeResponse - response to an Invoke request -->
1757
1758 <xs:element name="InvokeResponse" type="InvokeResponseType"/>
1759
1760 <xs:complexType name="InvokeResponseType">
1761   <xs:complexContent>
1762     <xs:extension base="ResponseAbstractType">
1763       <xs:sequence>
1764         <xs:element ref="InvokeResponseItem" maxOccurs="unbounded" />
1765       </xs:sequence>
1766     </xs:extension>
1767   </xs:complexContent>
1768 </xs:complexType>
1769
1770 <!-- InvokeResponseItem - container for the result of each InvokeItem -->
1771
1772 <xs:element name="InvokeResponseItem" type="InvokeResponseItemType"/>
1773
1774 <xs:complexType name="InvokeResponseItemType">
1775   <xs:sequence>
1776     <xs:any namespace="##other"
1777       processContents="lax"
1778       minOccurs="0"
1779       maxOccurs="unbounded" />
1780   </xs:sequence>
1781   <xs:attribute name="ref" type="xs:string" use="required" />
1782 </xs:complexType>
1783
1784
```

```
1785 <!-- QueryRegistered - query for the registered service instances -->
1786
1787 <xs:element name="QueryRegistered" type="QueryRegisteredType" />
1788
1789 <xs:complexType name="QueryRegisteredType">
1790   <xs:complexContent>
1791     <xs:extension base="RequestAbstractType">
1792       <xs:sequence>
1793         <xs:element ref="ServiceHandle" minOccurs="0" maxOccurs="unbounded" />
1794       </xs:sequence>
1795     </xs:extension>
1796   </xs:complexContent>
1797 </xs:complexType>
1798
1799
1800 <!-- QueryRegisteredResponse - response for QueryRegistered request -->
1801
1802 <xs:element name="QueryRegisteredResponse" type="disco:QueryResponseType" />
1803
1804
1805 <!-- Register - request for a new hosted or proxied service instance -->
1806
1807 <xs:element name="Register" type="RegisterType" />
1808
1809 <xs:complexType name="RegisterType">
1810   <xs:complexContent>
1811     <xs:extension base="RequestAbstractType">
1812       <xs:sequence>
1813         <xs:element ref="wsa:EndpointReference" maxOccurs="unbounded" />
1814       </xs:sequence>
1815     </xs:extension>
1816   </xs:complexContent>
1817 </xs:complexType>
1818
1819
1820 <!-- RegisterResponse - response to the Register request -->
1821
1822 <xs:element name="RegisterResponse" type="RegisterResponseType" />
1823
1824 <xs:complexType name="RegisterResponseType">
1825   <xs:complexContent>
1826     <xs:extension base="ResponseAbstractType">
1827       <xs:sequence>
1828         <xs:element ref="RegisterResponseItem" maxOccurs="unbounded" />
1829       </xs:sequence>
1830     </xs:extension>
1831   </xs:complexContent>
1832 </xs:complexType>
1833
1834 <xs:element name="RegisterResponseItem" type="RegisterResponseItemType" />
1835
1836 <xs:complexType name="RegisterResponseItemType">
1837   <xs:sequence>
1838     <xs:element ref="ServiceHandle" />
1839   </xs:sequence>
1840   <xs:attribute name="ref" type="xs:string" use="required" />
1841 </xs:complexType>
1842
1843
1844 <!-- SetStatus - set the status of a hosted/proxied service instance -->
1845
1846 <xs:element name="SetStatus" type="SetStatusType" />
1847
1848 <xs:complexType name="SetStatusType">
1849   <xs:complexContent>
1850     <xs:extension base="RequestAbstractType">
1851       <xs:sequence>
```

```
1852         <xs:element ref="SetStatusItem" maxOccurs="unbounded" />
1853     </xs:sequence>
1854 </xs:extension>
1855 </xs:complexContent>
1856 </xs:complexType>
1857
1858 <xs:element name="SetStatusItem" type="SetStatusItemType"/>
1859
1860 <xs:complexType name="SetStatusItemType">
1861     <xs:sequence>
1862         <xs:element ref="ServiceStatus" />
1863         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1864     </xs:sequence>
1865     <xs:attribute name="itemID" type="xs:string" use="required" />
1866 </xs:complexType>
1867
1868
1869 <!-- SetStatusResponse - response to the SetStatus request -->
1870
1871 <xs:element name="SetStatusResponse" type="SetStatusResponseType"/>
1872
1873 <xs:complexType name="SetStatusResponseType">
1874     <xs:complexContent>
1875         <xs:extension base="ResponseAbstractType" />
1876     </xs:complexContent>
1877 </xs:complexType>
1878
1879
1880 <!-- Update - update the configuration of the hosted/proxied service instance -->
1881
1882 <xs:element name="Update" type="UpdateType"/>
1883
1884 <xs:complexType name="UpdateType">
1885     <xs:complexContent>
1886         <xs:extension base="RequestAbstractType">
1887             <xs:sequence>
1888                 <xs:element ref="UpdateItem" maxOccurs="unbounded" />
1889             </xs:sequence>
1890         </xs:extension>
1891     </xs:complexContent>
1892 </xs:complexType>
1893
1894 <xs:element name="UpdateItem" type="UpdateItemType" />
1895
1896 <xs:complexType name="UpdateItemType">
1897     <xs:sequence>
1898         <xs:element ref="ServiceHandle" />
1899         <xs:element ref="wsa:EndpointReference" />
1900     </xs:sequence>
1901     <xs:attribute name="itemID" type="xs:string" use="required" />
1902 </xs:complexType>
1903
1904
1905 <!-- UpdateResponse - the response to the Update request -->
1906
1907 <xs:element name="UpdateResponse" type="UpdateResponseType"/>
1908
1909 <xs:complexType name="UpdateResponseType">
1910     <xs:complexContent>
1911         <xs:extension base="ResponseAbstractType" />
1912     </xs:complexContent>
1913 </xs:complexType>
1914
1915
1916 <!-- Poll - Poll for new service requests -->
1917
1918 <xs:element name="Poll" type="dp:PollType"/>
```

```

1919
1920
1921 <!--PollResponse - response for the Poll request -->
1922
1923 <xs:element name="PollResponse" type="dp:PollResponseType"/>
1924
1925
1926 <!-- ProxyInvoke - proxied invocation of CSI -->
1927
1928 <xs:element name="ProxyInvoke" type="ProxyInvokeType"/>
1929
1930 <xs:complexType name="ProxyInvokeType">
1931   <xs:complexContent>
1932     <xs:extension base="RequestAbstractType">
1933       <xs:sequence>
1934         <xs:element ref="ProxyInvokeItem" maxOccurs="unbounded" />
1935       </xs:sequence>
1936     </xs:extension>
1937   </xs:complexContent>
1938 </xs:complexType>
1939
1940
1941 <!-- Declaration of ProxyInvokeItem element -->
1942 <xs:element name="ProxyInvokeItem" type="ProxyInvokeItemType"/>
1943
1944 <xs:complexType name="ProxyInvokeItemType">
1945   <xs:sequence>
1946     <xs:element ref="ServiceHandle" />
1947     <xs:element ref="InvocationContext" />
1948     <xs:element ref="RequestHeaders" minOccurs="0" />
1949     <xs:any namespace="##other"
1950       processContents="lax"
1951       minOccurs="0"
1952       maxOccurs="unbounded" />
1953   </xs:sequence>
1954   <xs:attribute name="itemID" type="xs:string" use="required" />
1955   <xs:anyAttribute namespace="##other" processContents="lax"/>
1956 </xs:complexType>
1957
1958 <xs:element name="RequestHeaders" />
1959
1960
1961 <!-- ProxyInvokeResponse - response to the ProxyInvoke request -->
1962
1963 <xs:element name="ProxyInvokeResponse" type="ProxyInvokeResponseType"/>
1964
1965 <xs:complexType name="ProxyInvokeResponseType">
1966   <xs:complexContent>
1967     <xs:extension base="ResponseAbstractType">
1968       <xs:sequence>
1969         <xs:element ref="ProxyInvokeResponseItem" minOccurs="0"
1970           maxOccurs="unbounded" />
1971       </xs:sequence>
1972     </xs:extension>
1973   </xs:complexContent>
1974 </xs:complexType>
1975
1976 <!-- ProxyInvokeResponseItem - container for each service level invocation -->
1977
1978 <xs:element name="ProxyInvokeResponseItem" type="ProxyInvokeResponseItemType"/>
1979
1980 <xs:complexType name="ProxyInvokeResponseItemType">
1981   <xs:sequence>
1982     <xs:element ref="ServiceHandle" />
1983     <xs:element ref="ResponseHeaders" minOccurs="0" />
1984     <xs:any namespace="##other"
1985       processContents="lax"
    
```

```
1986         minOccurs="0"  
1987         maxOccurs="unbounded" />  
1988     </xs:sequence>  
1989     <xs:attribute name="ref" type="xs:string" use="required" />  
1990 </xs:complexType>  
1991  
1992 <xs:element name="ResponseHeaders" />  
1993  
1994  
1995  
1996 <!-- End of Interface Definitions } -->  
1997  
1998 </xs:schema>  
1999
```

6. Service Hosting/Proxying Service WSDL

2000

```
2001 <?xml version="1.0"?>
2002 <definitions name="shps-svc"
2003   targetNamespace="urn:liberty:shps:2006-12"
2004   xmlns:tns="urn:liberty:shps:2006-12"
2005   xmlns="http://schemas.xmlsoap.org/wsdl/"
2006   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2007   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2008   xmlns:sb="urn:liberty:sb:2006-12"
2009   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2010   xmlns:shps="urn:liberty:shps:2006-12"
2011   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2012   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2013     http://schemas.xmlsoap.org/wsdl/
2014     http://www.w3.org/2006/02/addressing/wsdl
2015     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2016
2017   <types>
2018     <xsd:schema>
2019       <xsd:import namespace="urn:liberty:shps:2006-12"
2020         schemaLocation="liberty-idwsf-shps-v1.0.xsd"/>
2021     </xsd:schema>
2022   </types>
2023
2024   <message name="Query">
2025     <part name="body" element="shps:Query"/>
2026   </message>
2027   <message name="QueryResponse">
2028     <part name="body" element="shps:QueryResponse"/>
2029   </message>
2030
2031   <message name="QueryRegistered">
2032     <part name="body" element="shps:QueryRegistered"/>
2033   </message>
2034   <message name="QueryRegisteredResponse">
2035     <part name="body" element="shps:QueryRegisteredResponse"/>
2036   </message>
2037
2038   <message name="Register">
2039     <part name="body" element="shps:Register"/>
2040   </message>
2041   <message name="RegisterResponse">
2042     <part name="body" element="shps:RegisterResponse"/>
2043   </message>
2044
2045   <message name="Update">
2046     <part name="body" element="shps:Update"/>
2047   </message>
2048   <message name="UpdateResponse">
2049     <part name="body" element="shps:UpdateResponse"/>
2050   </message>
2051
2052   <message name="Delete">
2053     <part name="body" element="shps>Delete"/>
2054   </message>
2055   <message name="DeleteResponse">
2056     <part name="body" element="shps>DeleteResponse"/>
2057   </message>
2058
2059   <message name="Invoke">
2060     <part name="body" element="shps:Invoke"/>
2061   </message>
2062   <message name="InvokeResponse">
2063     <part name="body" element="shps:InvokeResponse"/>
2064   </message>
2065
```

```
2066 <message name="GetStatus">
2067   <part name="body" element="shps:GetStatus" />
2068 </message>
2069 <message name="GetStatusResponse">
2070   <part name="body" element="shps:GetStatusResponse" />
2071 </message>
2072
2073 <message name="SetStatus">
2074   <part name="body" element="shps:SetStatus" />
2075 </message>
2076 <message name="SetStatusResponse">
2077   <part name="body" element="shps:SetStatusResponse" />
2078 </message>
2079
2080 <portType name="SHPSPort">
2081
2082   <operation name="Query">
2083     <input message="tns:Query"
2084       wsaw:Action="urn:liberty:shps:2006-12:Query" />
2085     <output message="tns:QueryResponse"
2086       wsaw:Action="urn:liberty:shps:2006-12:QueryResponse" />
2087   </operation>
2088
2089   <operation name="QueryRegistered">
2090     <input message="tns:QueryRegistered"
2091       wsaw:Action="urn:liberty:shps:2006-12:QueryRegistered" />
2092     <output message="tns:QueryRegisteredResponse"
2093       wsaw:Action="urn:liberty:shps:2006-12:QueryRegisteredResponse" />
2094   </operation>
2095
2096   <operation name="Register">
2097     <input message="tns:Register"
2098       wsaw:Action="urn:liberty:shps:2006-12:Register" />
2099     <output message="tns:RegisterResponse"
2100       wsaw:Action="urn:liberty:shps:2006-12:RegisterResponse" />
2101   </operation>
2102
2103   <operation name="Update">
2104     <input message="tns:Update"
2105       wsaw:Action="urn:liberty:shps:2006-12:Update" />
2106     <output message="tns:UpdateResponse"
2107       wsaw:Action="urn:liberty:shps:2006-12:UpdateResponse" />
2108   </operation>
2109
2110   <operation name="Delete">
2111     <input message="tns>Delete"
2112       wsaw:Action="urn:liberty:shps:2006-12>Delete" />
2113     <output message="tns>DeleteResponse"
2114       wsaw:Action="urn:liberty:shps:2006-12>DeleteResponse" />
2115   </operation>
2116
2117   <operation name="Invoke">
2118     <input message="tns:Invoke"
2119       wsaw:Action="urn:liberty:shps:2006-12:Invoke" />
2120     <output message="tns:InvokeResponse"
2121       wsaw:Action="urn:liberty:shps:2006-12:InvokeResponse" />
2122   </operation>
2123
2124   <operation name="GetStatus">
2125     <input message="tns:GetStatus"
2126       wsaw:Action="urn:liberty:shps:2006-12:GetStatus" />
2127     <output message="tns:GetStatusResponse"
2128       wsaw:Action="urn:liberty:shps:2006-12:GetStatusResponse" />
2129   </operation>
2130
2131   <operation name="SetStatus">
2132     <input message="tns:SetStatus"
```

```
2133     wsaw:Action="urn:liberty:shps:2006-12:SetStatus" />
2134     <output message="tns:SetStatusResponse"
2135     wsaw:Action="urn:liberty:shps:2006-12:SetStatusResponse" />
2136   </operation>
2137
2138 </portType>
2139
2140 <!--
2141 An example of a binding and service that can be used with this
2142 abstract service description is shpsided below.
2143 -->
2144
2145 <binding name="SHPSBinding" type="tns:SHPSPort">
2146
2147   <soap:binding style="document"
2148     transport="http://schemas.xmlsoap.org/soap/http"/>
2149
2150   <operation name="Query">
2151     <input> <soap:body use="literal"/> </input>
2152     <output> <soap:body use="literal"/> </output>
2153   </operation>
2154
2155   <operation name="QueryRegistered">
2156     <input> <soap:body use="literal"/> </input>
2157     <output> <soap:body use="literal"/> </output>
2158   </operation>
2159
2160   <operation name="Register">
2161     <input> <soap:body use="literal"/> </input>
2162     <output> <soap:body use="literal"/> </output>
2163   </operation>
2164
2165   <operation name="Update">
2166     <input> <soap:body use="literal"/> </input>
2167     <output> <soap:body use="literal"/> </output>
2168   </operation>
2169
2170   <operation name="Delete">
2171     <input> <soap:body use="literal"/> </input>
2172     <output> <soap:body use="literal"/> </output>
2173   </operation>
2174
2175   <operation name="Invoke">
2176     <input> <soap:body use="literal"/> </input>
2177     <output> <soap:body use="literal"/> </output>
2178   </operation>
2179
2180   <operation name="GetStatus">
2181     <input> <soap:body use="literal"/> </input>
2182     <output> <soap:body use="literal"/> </output>
2183   </operation>
2184
2185   <operation name="SetStatus">
2186     <input> <soap:body use="literal"/> </input>
2187     <output> <soap:body use="literal"/> </output>
2188   </operation>
2189 </binding>
2190
2191 <service name="SHPService">
2192
2193   <port name="SHPSPort" binding="tns:SHPSBinding">
2194
2195     <!-- Modify with the REAL SOAP endpoint -->
2196
2197     <soap:address location="http://example.com/shps"/>
2198
2199
```

```
2200     </port>
2201
2202     </service>
2203
2204 </definitions>
2205
```

References

Normative

2206

2207

2208 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-
2209 errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>

2210 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"
2211 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>

2212 [LibertyDP] Cahill, Conor P., eds. "Liberty ID-WSF Design Patterns Specification," Version 1.0-05, Liberty Alliance
2213 Project (07 May, 2007). <http://www.projectliberty.org/specs>

2214 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-
2215 errata-v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>

2216 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version
2217 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>

2218 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14
2219 December, 2004). <http://www.projectliberty.org/specs>

2220 [LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,
2221 Single Sign-On, and Identity Mapping Services Specification ," Version 2.0-errata-v1.0, Liberty Alliance
2222 Project (28 November, 2006). <http://www.projectliberty.org/specs>

2223 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty
2224 ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007).
2225 <http://www.projectliberty.org/specs>

2226 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
2227 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>

2228 [RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251,
2229 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2251.txt>

2230 [RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A.
2231 Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force
2232 <http://www.ietf.org/rfc/rfc2252.txt>

2233 [RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search
2234 Results," RFC 2891, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2891.txt>

2235 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Pro-
2236 tocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS
2237 Standard, Organization for the Advancement of Structured Information Standards [http://www.oasis-
2238 open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf](http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)

2239 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
2240 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
2241 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-
2242 open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf)

2243 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
2244 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"

- 2245 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards
2246 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- 2247 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
2248 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
2249 <http://www.w3.org/TR/xmlschema-1/>
- 2250 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman,
2251 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
2252 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 2253 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
2254 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
2255 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 2256 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
2257 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 2258 [XPath] Clark, J., DeRose, S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0,"
2259 Recommendation, W3C <http://www.w3.org/TR/xpath> [August 2003].

2260 Informative

- 2261 [LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty
2262 Alliance Project (13 January, 2005). <http://www.projectliberty.org/specs>
- 2263 [LibertyIDPPGuide] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Implementation
2264 Guidelines," Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 2265 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework
2266 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>