



# Liberty ID-WSF Service Hosting and Proxying Service Specification

Version: 1.0

**Editors:**

Conor Cahill, Intel Corporation

**Contributors:**

Shelagh Callahan, Intel Corporation

Jane Dashevsky, Intel Corporation

Hubert Le Van Gong, Sun

Andrew Lindsay-Stewart, Vodafone

Tapio Kaukonen, Nokia

Sampo Kellomäki, Symlabs, Inc.

Paul Madsen, NTT

Paul Miller, Gemplus

Pierre Vannel, Gemplus

Sean Walker, Axalto

George Fletcher, AOL LLC

Hiroyoshi Takiguchi, NTT

Greg Whitehead, Hewlett-Packard

**Abstract:**

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

**Filename:** liberty-idwsf-shps-v1.0.pdf

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

35 Liberty Alliance Project  
36 Licensing Administrator  
37 c/o IEEE-ISTO  
38 445 Hoes Lane  
39 Piscataway, NJ 08855-1331, USA  
40 info@projectliberty.org

## 41 Contents

42	1. Introduction	5
43	1.1. Notation and Conventions	5
44	1.1.1. XML Namespaces	5
45	2. Overview	6
46	2.1. Client Service Components	6
47	3. Service Hosting or Proxying Service Definition	8
48	3.1. Service URIs	8
49	3.2. Data Definitions	8
50	3.2.1. Status Codes	8
51	3.2.2. Request and Response Abstract Types	9
52	3.2.3. Service Mode	10
53	3.2.4. Service Handle	10
54	3.2.5. InvocationContext	11
55	3.2.6. Callback EPR	12
56	3.2.7. Service Descriptor	12
57	3.3. Operation: <i>Query</i>	14
58	3.3.1. wsa:Action values for Query Messages	14
59	3.3.2. Query Message	14
60	3.3.3. QueryResponse Message	15
61	3.3.4. Query Processing Rules	16
62	3.4. Operation: <i>QueryRegistered</i>	16
63	3.4.1. wsa:Action values for QueryRegistered Messages	17
64	3.4.2. QueryRegistered Message	17
65	3.4.3. QueryRegisteredResponse Message	17
66	3.4.4. QueryRegistered Processing Rules	18
67	3.5. Operation: <i>Register</i>	18
68	3.5.1. wsa:Action values for Register Messages	19
69	3.5.2. Register Message	19
70	3.5.3. RegisterResponse Message	20
71	3.5.4. Register Processing Rules	21
72	3.6. Operation: <i>Update</i>	22
73	3.6.1. wsa:Action values for Update Messages	22
74	3.6.2. Update Message	22
75	3.6.3. UpdateResponse Message	24
76	3.6.4. Update Processing Rules	25
77	3.7. Operation: <i>Delete</i>	25
78	3.7.1. wsa:Action values for Delete Messages	25
79	3.7.2. Delete Message	26
80	3.7.3. DeleteResponse Message	26
81	3.7.4. Delete Processing Rules	27
82	3.8. Operation: <i>Invoke</i>	27
83	3.8.1. wsa:Action values for Invoke Messages	27
84	3.8.2. Invoke Message	28
85	3.8.3. InvokeResponse Message	29
86	3.8.4. Invoke Processing Rules	30
87	3.9. Operation: <i>GetStatus</i>	31
88	3.9.1. wsa:Action values for GetStatus Messages	31
89	3.9.2. GetStatus Message	31
90	3.9.3. GetStatusResponse Message	32
91	3.9.4. GetStatus Processing Rules	33
92	3.10. Operation: <i>SetStatus</i>	34
93	3.10.1. wsa:Action values for SetStatus Messages	34

94	3.10.2. <i>SetStatus</i> Message .....	34
95	3.10.3. <i>SetStatusResponse</i> Message .....	35
96	3.10.4. <i>SetStatus</i> Processing Rules .....	36
97	3.11. Operation: <i>Poll</i> .....	36
98	3.11.1. <i>wsa:Action</i> values for <i>Poll</i> Messages .....	36
99	3.11.2. <i>Poll</i> Message .....	36
100	3.11.3. <i>PollResponse</i> Message .....	37
101	3.11.4. <i>Poll</i> Processing Rules .....	39
102	3.12. Operation: <i>ProxyInvoke</i> .....	39
103	3.12.1. <i>wsa:Action</i> values for <i>ProxyInvoke</i> Messages .....	39
104	3.12.2. <i>ProxyInvoke</i> Message .....	39
105	3.12.3. <i>ProxyInvokeResponse</i> Message .....	41
106	3.12.4. <i>ProxyInvoke</i> Processing Rules .....	43
107	4. Service Hosting/Proxying Service Schema .....	44
108	5. Service Hosting/Proxying Service WSDL .....	51
109	References .....	55

## 110 **1. Introduction**

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

### 117 **1.1. Notation and Conventions**

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

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

122 These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application  
123 features and behavior that affect the interoperability and security of implementations. When these words are not  
124 capitalized, they are meant in their natural-language sense.

#### 125 **1.1.1. XML Namespaces**

126 The following XML namespaces are referred to in this document:

127 • The prefix *shps*: represents the Service Hosting and Proxying Service namespace. This namespace is the default  
128 for instance fragments, type names, and element names in this document. In schema listings, and in examples of  
129 SHPS messages and fragments thereof, this is the default namespace when no prefix is shown:

130 *urn:liberty:shps:2007-09*

131 • The prefix *saml2*: stands for the SAMLv2 assertion namespace [\[SAMLCore2\]](#):

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

133 • The prefix *samlp2*: stands for the SAMLv2 protocol namespace [\[SAMLCore2\]](#):

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

135 • The prefix *xs*: stands for the W3C XML schema namespace [\[Schema1-2\]](#):

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

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

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

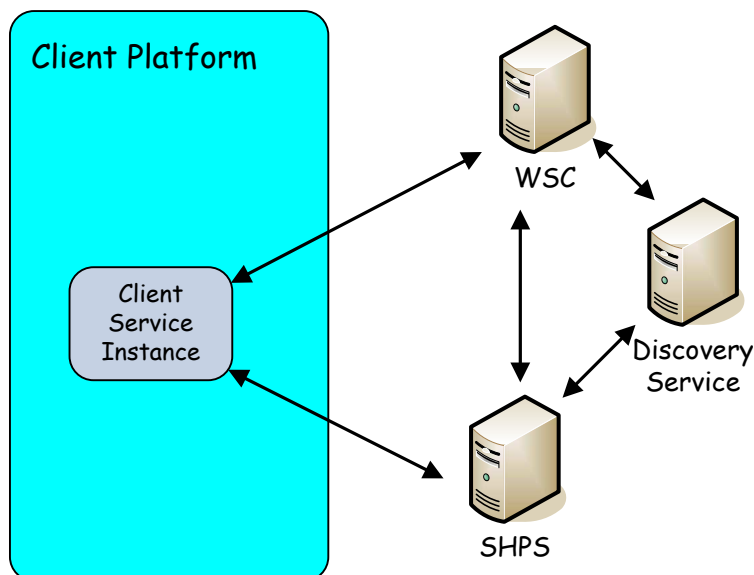
## 139 2. Overview

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

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

### 145 2.1. Client Service Components

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



147

148

**Figure 1. CSI Actors**

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

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

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

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

- 159
- Clients frequently have limited communications bandwidth (when compared to online services).
  - Clients have more tenuous connectivity (being unavailable when the user goes through a tunnel or turns the device off for the night).
- 160  
161

- 162 • Clients are frequently behind network firewalls, preventing incoming service invocation without modifications to  
163 the firewall rules.
- 164 • Multiple providers talking to the same service endpoint on the client for a particular user's service can use that  
165 service endpoint as a correlation handle for the user and potentially collude without user knowledge or control.
- 166 The user, and their client service instance, have the choice of the following hosting solutions:
- 167 • **Stand-alone** - the service is hosted exclusively on the client device and all service invokers must communicate  
168 directly with the device. This is the standard ID-WSF web services provider model where the service instance  
169 maintains a service metadata description at the Liberty ID-WSF Discovery Service (DS). Web service consumers  
170 (WSCs) discover and invoke the service instance using the Discovery Service.
- 171 Such implementations must deal with or accept the connectivity and privacy issues outlined above. The PAOS  
172 protocol (see [[LibertyPAOS](#)]) may be used with stand-alone service instances to resolve some of the connectivity  
173 issues.
- 174 In most cases the stand-alone solution is used where collusion protection is not a concern and the service instance  
175 is on an always-connected device exposed directly on an external network.
- 176 • **Proxied** - the service instance is hosted exclusively on the client device, but uses the Liberty ID-WSF Service  
177 Hosting/Proxying service (SHPS) to proxy incoming calls. This solves two of the problems listed above: a) the  
178 privacy breaking cross-provider collusion concern is mitigated by the large number of client using the same SHPS  
179 service and b) the client doesn't have to have an externally exposed interface as it can poll the SHPS service for  
180 incoming request.
- 181 In this case, the SHPS would be registered as the endpoint for the service instance for the user in the DS. WSCs  
182 would invoke the service instance at the SHPS and the SHPS would forward the request to the client, get the  
183 response back and forward the response to the WSC. The WSC would not be aware that the proxying is taking  
184 place.
- 185 If the service proxy hosted at SHPS is invoked when the client instance is not available, the call fails as in this  
186 mode the SHPS is not configured to act in the name of the client.
- 187 • **Hosted** - a mirror of the service instance is hosted on the SHPS. Requests for service are handled directly by the  
188 hosted instance without additional interaction with the client instance. The client service instance keeps the  
189 hosted mirror service instance up-to-date as necessary.
- 190 In this case, the SHPS would be registered as the endpoint for the service instance for the user in the DS. WSCs  
191 would invoke the service instance at the SHPS and SHPS would respond directly without involving the client.
- 192 • **Proxied+Hosted** - both hosting and proxying are implemented. Proxying is used when the client is available  
193 and when the client is not available SHPS is able to respond to the request using the data in it's mirrored service  
194 instance.

195 **3. Service Hosting or Proxying Service Definition**

196 A concrete definition of the SHPS interfaces.

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

199 The complete schema for the SHPS is included in this document, see [Section 4: Service Hosting/Proxying Service](#)  
 200 Schema .

201 **3.1. Service URIs**

202 **Table 1. SHPS URIs**

Use	URI
Service Type	<i>urn:liberty:shps:2007-09</i>
Query wsa:Action	<i>urn:liberty:shps:2007-09:Query</i>
QueryResponse wsa:Action	<i>urn:liberty:shps:2007-09:QueryResponse</i>
QueryRegistered wsa:Action	<i>urn:liberty:shps:2007-09:QueryRegistered</i>
QueryRegisteredResponse wsa:Action	<i>urn:liberty:shps:2007-09:QueryRegisteredResponse</i>
Register wsa:Action	<i>urn:liberty:shps:2007-09:Register</i>
RegisterResponse wsa:Action	<i>urn:liberty:shps:2007-09:RegisterResponse</i>
Update wsa:Action	<i>urn:liberty:shps:2007-09:Update</i>
UpdateResponse wsa:Action	<i>urn:liberty:shps:2007-09:UpdateResponse</i>
Delete wsa:Action	<i>urn:liberty:shps:2007-09&gt;Delete</i>
DeleteResponse wsa:Action	<i>urn:liberty:shps:2007-09&gt;DeleteResponse</i>
Invoke wsa:Action	<i>urn:liberty:shps:2007-09:Invoke</i>
InvokeResponse wsa:Action	<i>urn:liberty:shps:2007-09:InvokeResponse</i>
GetStatus wsa:Action	<i>urn:liberty:shps:2007-09:GetStatus</i>
GetStatusResponse wsa:Action	<i>urn:liberty:shps:2007-09:GetStatusResponse</i>
SetStatus wsa:Action	<i>urn:liberty:shps:2007-09:SetStatus</i>
SetStatusResponse wsa:Action	<i>urn:liberty:shps:2007-09:SetStatusResponse</i>
Poll wsa:Action	<i>urn:liberty:shps:2007-09:Poll</i>
PollResponse wsa:Action	<i>urn:liberty:shps:2007-09:PollResponse</i>
ProxyInvoke wsa:Action	<i>urn:liberty:shps:2007-09:ProxyInvoke</i>
ProxyInvokeResponse wsa:Action	<i>urn:liberty:shps:2007-09:ProxyInvokeResponse</i>

203 **3.2. Data Definitions**

204 **3.2.1. Status Codes**



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

- 211 • Failed
- 212 • Partial
- 213 • OK
- 214 • NeedServiceData
- 215 • NoResults
- 216 • NoSuchService

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

## 222 3.2.2. Request and Response Abstract Types

### 223 3.2.2.1. Complex Type RequestAbstractType

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

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

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

```
229 <!-- RequestAbstractType - common request message structure -->  
230  
231 <xs:complexType name="RequestAbstractType" abstract="true">  
232   <xs:anyAttribute namespace="##other" processContents="lax"/>  
233 </xs:complexType>  
234
```

### 235 3.2.2.2. Complex Type ResponseAbstractType

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

- 238 • `<lu:Status>` [**Required**] - The `<lu:Status>` element is used to convey status codes and related information.  
239 The schema fragment is defined in the Liberty ID-WSF Utility schema. The local definition of status codes are  
240 described in [Section 3.2.1](#).
- 241 • `anyAttribute` [**Optional**] - An attribute from a namespace other than that of this specification.

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

```
243 <!-- ResponseAbstractType - common message response structure -->
244
245 <xs:complexType name="ResponseAbstractType" abstract="true">
246   <xs:sequence>
247     <xs:element ref="lu:Status"/>
248   </xs:sequence>
249   <xs:anyAttribute namespace="##other" processContents="lax"/>
250 </xs:complexType>
251
```

### 252 3.2.3. Service Mode

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

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

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

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

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

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

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

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

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

### 270 3.2.4. Service Handle

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

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

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

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

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

```
284 <!-- ServiceHandle - a reference to a hosted/proxied service instance -->
285
286 <xs:element name="ServiceHandle" type="xs:anyURI"/>
287
```

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

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

```
290 <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
291
```

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

### 293 3.2.5. InvocationContext

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

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

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

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

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

```
306 <!-- InvocationContext - how the proxied service instance was invoked -->
307
308 <xs:element name="InvocationContext" type="InvocationContextType" />
309
310 <xs:complexType name="InvocationContextType">
311   <xs:sequence>
312     <xs:element ref="InvokingProvider" />
313     <xs:element ref="InvokingPrincipal" minOccurs="0" />
314     <xs:element ref="disco:SecurityMechID" />
315   </xs:sequence>
316   <xs:anyAttribute namespace="##other" processContents="lax"/>
317 </xs:complexType>
318
319 <xs:element name="InvokingProvider" type="xs:string" />
320 <xs:element name="InvokingPrincipal" type="saml2:NameIDType" />
321
```

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

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

```
324 <shps:InvocationContext>
325   <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
326   <shps:InvokingPrincipal
327     Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
328     NameQualifier="http://idps-r-us.com" >
329     uuid:23923-023843-230932-230923
330   </shps:InvokingPrincipal>
331   <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
332 </shps:InvocationContext>
333
```

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

### 335 3.2.6. Callback EPR

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

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

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

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

```
343 <!-- CallbackEPR - where the CSI can receive ProxyInvoke requests -->
344
345 <xs:element name="CallbackEPR" type="wsa:EndpointReferenceType"/>
346
```

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

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

```
349 <shps:CallbackEPR>
350   <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
351 </shps:CallbackEPR>
352
```

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

### 354 3.2.7. Service Descriptor

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

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

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

```
359 <!-- ServiceMode - the proxied or hosted mode of the service instance -->
360
361 <xs:element name="ServiceMode" type="xs:anyURI"/>
362
```

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

365 • A new element `<shps:ServiceStatus>` is defined with the following schema:

```
366 <!-- ServiceStatus - the enabled/disabled status of the service instance -->  
367  
368 <xs:element name="ServiceStatus" type="xs:anyURI"/>  
369
```

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

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

```
373 <!-- ServiceHandle - a reference to a hosted/proxied service instance -->  
374  
375 <xs:element name="ServiceHandle" type="xs:anyURI"/>  
376
```

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

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

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

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

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

389 • The `<disco:Abstract>` and `<disco:ProviderID>` elements MAY be absent.

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

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

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

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

404 An example `<shps:ServiceDescriptor>` is shown below.

```
405 <wsa:EndpointReference>
406   <wsa:Address>
407     http://www.w3.org/2005/08/addressing/anonymous
408   </wsa:Address>
409
410   <wsa:Metadata>
411     <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
412     <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
413     <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
414     <disco:ServiceType>urn:liberty:ps:2007-11</disco:ServiceType>
415     <disco:Framework version="2.0" />
416     <disco:SecurityContext>
417       <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
418       <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</disco:SecurityMechID>
419     </disco:SecurityContext>
420   </wsa:Metadata>
421 </wsa:EndpointReference>
422
```

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

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

### 427 **3.3. Operation: *Query***

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

#### 430 **3.3.1. `wsa:Action` values for `query` Messages**

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

#### 434 **3.3.2. *Query* Message**

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

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

- 439 • Zero or more `<shps:ServiceMode>` elements MAY be present within the `<disco:RequestedServiceType>`  
440 element. If present, this indicates the type of service mode requested by the client.
- 441 Multiple `<shps:ServiceMode>` elements may be specified and any Service Descriptor that matches any of the  
442 specified `<shps:ServiceMode>` values is to be considered a match.

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

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

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

```
449 <!-- Query - query for ability to host or proxy services -->  
450  
451 <xs:element name="Query" type="disco:QueryType"/>  
452
```

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

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

```
457 <shps:Query>  
458 <disco:RequestedService>  
459 <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>  
460 <disco:SecurityMechID>urn:liberty:sm:2006-08:TLS:SAMLV2</disco:SecurityMechID>  
461 <disco:Framework version="2.0" />  
462 <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>  
463 </disco:RequestedService>  
464 </shps:Query>  
465
```

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

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

```
469 <shps:Query>  
470 <disco:RequestedService>  
471 <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>  
472 <disco:SecurityMechID>urn:liberty:sm:2006-08:TLS:SAMLV2</disco:SecurityMechID>  
473 <disco:Framework version="2.0" />  
474 <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>  
475 </disco:RequestedService>  
476 </shps:Query>  
477
```

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

### 479 **3.3.3. QueryResponse Message**

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

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

```

485 <!--QueryResponse - response for the Query request -->
486
487 <xs:element name="QueryResponse" type="disco:QueryResponseType"/>
488
    
```

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

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

```

492 <shps:QueryResponse>
493 <lu:Status code="OK" />
494 <wsa:EndpointReference>
495 <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
496 <wsa:Metadata>
497 <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
498 <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
499 <disco:Framework version="2.0" />
500 <disco:SecurityContext>
501 <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
502 </disco:SecurityContext>
503 </wsa:Metadata>
504 </wsa:EndpointReference>
505 </shps:QueryResponse>
506
    
```

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

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

```

510 <shps:QueryResponse>
511 <lu:Status code="OK" />
512 <wsa:EndpointReference>
513 <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
514 <wsa:Metadata>
515 <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
516 <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
517 <disco:Framework version="2.0" />
518 <disco:SecurityContext>
519 <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
520 </disco:SecurityContext>
521 </wsa:Metadata>
522 </wsa:EndpointReference>
523 </shps:QueryResponse>
524
    
```

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

### 526 3.3.4. Query Processing Rules

- 527 • The SHPS SHOULD return Service Descriptions for each service that it is willing to host or proxy which meets  
 528 the conditions in the request.
- 529 • The SHPS MAY use any factor at its disposal in deciding whether it is willing to host or proxy a service including  
 530 the identity of the user and/or the Advanced Client itself.
- 531 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code  
 532 MUST be "Failed"



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

### 535 **3.4. Operation: *QueryRegistered***

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

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

#### 540 **3.4.1. *wsa:Action* values for *QueryRegistered* Messages**

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

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

#### 545 **3.4.2. *QueryRegistered* Message**

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

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

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

```
551 <!-- QueryRegistered - query for the registered service instances -->
552
553 <xs:element name="QueryRegistered" type="QueryRegisteredType"/>
554
555 <xs:complexType name="QueryRegisteredType">
556   <xs:complexContent>
557     <xs:extension base="RequestAbstractType">
558       <xs:sequence>
559         <xs:element ref="ServiceHandle" minOccurs="0" maxOccurs="unbounded" />
560       </xs:sequence>
561     </xs:extension>
562   </xs:complexContent>
563 </xs:complexType>
564
```

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

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

```
568 <shps:QueryRegistered>
569   <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
570   <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
571 </shps:QueryRegistered>
572
```

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

#### 574 **3.4.3. *QueryRegisteredResponse* Message**

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

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

```
580 <!-- QueryRegisteredResponse - response for QueryRegistered request -->
581
582 <xs:element name="QueryRegisteredResponse" type="disco:QueryResponseType"/>
583
```

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

```
585 <shps:QueryResponse>
586 <lu:Status code="OK" />
587 <wsa:EndpointReference>
588 <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
589 <wsa:Metadata>
590 <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
591 <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
592 <shps:ServiceStatus>urn:liberty:shps:2007-09:status:enabled</shps:ServiceStatus>
593 <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
594 <disco:Framework version="2.0" />
595 <disco:SecurityContext>
596 <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:SAML</disco:SecurityMechID>
597 </disco:SecurityContext>
598 </wsa:Metadata>
599 </wsa:EndpointReference>
600 </shps:QueryResponse>
601
```

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

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

### 605 3.4.4. QueryRegistered Processing Rules

- 606 • The SHPS MUST only consider service instances registered by this Advanced Client in responding to the request.  
607 Service instances registered by other Advanced Clients MUST NOT be visible to this Advanced Client. There  
608 MUST be no difference in the response to a nonexistent Service Handle and a Service Handle that refers to a  
609 service instance registered by a different Advanced Client.
- 610 • The SHPS SHOULD return Service Descriptions for each registered service instance matching the specified  
611 Service Handle(s). If no Service Handles are specified, Service Descriptors for all of the registered service  
612 instances should be returned.
- 613 • If a specified Service Handle is not currently registered the SHPS SHOULD ignore it and only return the successful  
614 matches.
- 615 • If there were no successful matches of Service Handles, the top-level status code MUST be *Failed* and NO ID-  
616 WSF EPRs are to be returned. In such cases, if the SHPS is providing second level status codes, the second-level  
617 error code MUST be *NoResults*.
- 618 • If there are successful matches, the top-level status code MUST be "OK" and the successfully matching Service  
619 Descriptors are returned.

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

### 622 **3.5. Operation: *Register***

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

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

#### 628 **3.5.1. wsa:Action values for Register Messages**

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

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

#### 632 **3.5.2. Register Message**

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

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

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

```
637 <!-- Register - request for a new hosted or proxied service instance -->
638
639 <xs:element name="Register" type="RegisterType"/>
640
641 <xs:complexType name="RegisterType">
642   <xs:complexContent>
643     <xs:extension base="RequestAbstractType">
644       <xs:sequence>
645         <xs:element ref="wsa:EndpointReference" maxOccurs="unbounded" />
646       </xs:sequence>
647     </xs:extension>
648   </xs:complexContent>
649 </xs:complexType>
650
```

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

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

```

655 <shps:Register>
656   <wsa:EndpointReference lu:itemID="1">
657     <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
658     <wsa:Metadata>
659       <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
660       <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
661       <disco:Framework version="2.0" />
662       <disco:SecurityContext>
663         <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
664       </disco:SecurityContext>
665     </wsa:Metadata>
666   </wsa:EndpointReference>
667 </shps:Register>
668

```

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

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

```

674 <shps:Register>
675   <wsa:EndpointReference lu:itemID="1">
676     <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
677     <wsa:Metadata>
678       <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:proxied</shps:ServiceMode>
679       <disco:ServiceType>urn:liberty:id-sis-pp:2003-08</disco:ServiceType>
680       <disco:Framework version="2.0" />
681       <disco:SecurityContext>
682         <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
683       </disco:SecurityContext>
684       <shps:CallbackEPR>
685         <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
686       </shps:CallbackEPR>
687     </wsa:Metadata>
688   </wsa:EndpointReference>
689 </shps:Register>
690

```

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

### 692 3.5.3. RegisterResponse Message

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

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

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

```

705 <!-- RegisterResponse - response to the Register request -->
706
707 <xs:element name="RegisterResponse" type="RegisterResponseType"/>
708
709 <xs:complexType name="RegisterResponseType">
710   <xs:complexContent>
711     <xs:extension base="ResponseAbstractType">
712       <xs:sequence>
713         <xs:element ref="RegisterResponseItem" maxOccurs="unbounded" />
714       </xs:sequence>
715     </xs:extension>
716   </xs:complexContent>
717 </xs:complexType>
718
719 <xs:element name="RegisterResponseItem" type="RegisterResponseItemType" />
720
721 <xs:complexType name="RegisterResponseItemType">
722   <xs:sequence>
723     <xs:element ref="ServiceHandle" />
724   </xs:sequence>
725   <xs:attribute name="ref" type="xs:string" use="required" />
726 </xs:complexType>
727

```

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

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

```

730 <shps:RegisterResponse>
731   <lu:Status code="OK" />
732   <shps:RegisterResponseItem ref="1">
733     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023 </shps:ServiceHandle>
734   </shps:RegisterResponseItem>
735 </shps:RegisterResponse>
736

```

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

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

```

740 <shps:RegisterResponse>
741   <lu:Status code="OK" />
742   <shps:RegisterResponseItem ref="1">
743     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023 </shps:ServiceHandle>
744   </shps:RegisterResponseItem>
745   <shps:RegisterResponseItem ref="2">
746     <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238 927 </shps:ServiceHandle>
747   </shps:RegisterResponseItem>
748 </shps:RegisterResponse>
749

```

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

### 751 3.5.4. Register Processing Rules

- 752 • The `<shps:Register>` request MUST include an `itemID` attribute on each `<wsa:EndpointReference>`  
753 element in the request.
- 754 The value of the `itemID` attribute MUST be placed into the `ref` attribute in the `<shps:RegisterResponseItem>`  
755 element associated with the `<wsa:EndpointReference>` element in the request.
- 756 • The order of the elements in the response MAY be different than the order of the associated elements in the request.  
757 The `ref` attribute MUST be used to correlate the response item to its associated request element.
- 758 • The Advanced Client MAY alter a Service Description to have more restrictive settings than those returned from the  
759 SHPS in a `<shps:QueryResponse>`. For example, the Advanced Client may select a single security mechanism  
760 for its service instance at the SHPS, even though the SHPS is capable of supporting several different security  
761 mechanisms.
- 762 Any such alterations made SHOULD NOT include features or capabilities that the SHPSs has not expressed a  
763 willingness to expose.
- 764 • The SHPS MAY refuse to accept the registration of any Service Descriptor for which it is unable or unwilling to  
765 expose on behalf of the Advanced Client. In such cases the SHPS MUST return a failure.
- 766 • The Advanced Client MAY initiate a `<shps:Register>` request using a Service Descriptor that it created  
767 (rather than one obtained from the SHPS using the `<shps:Query>` interface). Of course, the SHPS MAY refuse  
768 such operations if it does not support exposing the requested service or any of the specific features requested.
- 769 • The SHPS MAY use any factor at its disposal in deciding whether or not to accept the registration (it MAY  
770 even deny a registration for a Service Descriptor that the SHPS previously returned to the Advanced Client in a  
771 `<shps:QueryResponse>` if the conditions for such willingness have changed).
- 772 • This operation MUST be atomic and if successful, all portions of the request MUST have succeeded. If any  
773 portion of the request fails, the entire request must fail. This requirement is mostly applicable in the case where  
774 the request includes multiple Service Descriptors.
- 775 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code  
776 MUST be "Failed"
- 777 • If the top-level status code is "Failed," the response MAY also contain *Forbidden* as a second-level status code.  
778 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will  
779 appear.

## 780 **3.6. Operation: Update**

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

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

### 785 **3.6.1. wsa:Action values for Update Messages**

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

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

### 789 **3.6.2. Update Message**

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

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

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

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

```
798 <!-- Update - update the configuration of the hosted/proxied service instance -->
799 <xs:element name="Update" type="UpdateType"/>
800 <xs:complexType name="UpdateType">
801   <xs:complexContent>
802     <xs:extension base="RequestAbstractType">
803       <xs:sequence>
804         <xs:element ref="UpdateItem" maxOccurs="unbounded" />
805       </xs:sequence>
806     </xs:extension>
807   </xs:complexContent>
808 </xs:complexType>
809 <xs:element name="UpdateItem" type="UpdateItemType" />
810 <xs:complexType name="UpdateItemType">
811   <xs:sequence>
812     <xs:element ref="ServiceHandle" />
813     <xs:element ref="wsa:EndpointReference" />
814   </xs:sequence>
815   <xs:attribute name="itemID" type="xs:string" use="required" />
816 </xs:complexType>
```

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

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

```

825 <shps:Update>
826   <shps:UpdateItem itemID="1">
827     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
828     <wsa:EndpointReference>
829       <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
830       <wsa:Metadata>
831         <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
832         <disco:ServiceType>urn:liberty:ps:2006-08</disco:ServiceType>
833         <disco:Framework version="2.0" />
834         <disco:SecurityContext>
835           <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
836           <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:Bearer</disco:SecurityMechID>
837         </disco:SecurityContext>
838       </wsa:Metadata>
839     </wsa:EndpointReference>
840   </shps:UpdateItem>
841   <shps:UpdateItem itemID="2">
842     <shps:ServiceHandle>uuid:38239-ad23fe-2938246-9238927</shps:ServiceHandle>
843     <wsa:EndpointReference>
844       <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
845       <wsa:Metadata>
846         <shps:ServiceMode>urn:liberty:shps:2007-09:svcmode:hosted</shps:ServiceMode>
847         <disco:ServiceType>urn:liberty:pp:2003-08</disco:ServiceType>
848         <disco:Framework version="2.0" />
849         <disco:SecurityContext>
850           <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
851         </disco:SecurityContext>
852       </wsa:Metadata>
853     </wsa:EndpointReference>
854   </shps:UpdateItem>
855 </shps:Update>
856

```

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

### 858 3.6.3. UpdateResponse Message

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

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

```

863 <!-- UpdateResponse - the response to the Update request -->
864
865 <xs:element name="UpdateResponse" type="UpdateResponseType"/>
866
867 <xs:complexType name="UpdateResponseType">
868   <xs:complexContent>
869     <xs:extension base="ResponseAbstractType" />
870   </xs:complexContent>
871 </xs:complexType>
872

```

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

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



```
876 <shps:UpdateResponse>
877   <lu:Status code="Partial">
878     <lu:Status code="NoSuchService" ref="2" />
879   </lu:Status>
880 </shps:UpdateResponse>
881
```

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

### 883 3.6.4. Update Processing Rules

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

## 910 **3.7. Operation: *Delete***

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

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

### 915 **3.7.1. wsa:Action values for Delete Messages**

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

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

### 919 **3.7.2. Delete Message**

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

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

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

```
924 <!-- Delete - delete (remove) a service instance -->
925
926 <xs:element name="Delete" type="DeleteType"/>
927
928 <xs:complexType name="DeleteType">
929   <xs:complexContent>
930     <xs:extension base="RequestAbstractType">
931       <xs:sequence>
932         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
933       </xs:sequence>
934     </xs:extension>
935   </xs:complexContent>
936 </xs:complexType>
937
```

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

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

```
941 <shps:Delete>
942   <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
943   <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
944 </shps:Delete>
945
```

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

### 947 **3.7.3. DeleteResponse Message**

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

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

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

```
952 <!-- DeleteResponse - response to a delete request -->
953
954 <xs:element name="DeleteResponse" type="DeleteResponseType"/>
955
956 <xs:complexType name="DeleteResponseType">
957   <xs:complexContent>
958     <xs:extension base="ResponseAbstractType" />
959   </xs:complexContent>
960 </xs:complexType>
961
```

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

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

```
965 <shps:DeleteResponse>
966   <lu:Status code="Partial">
967     <lu:Status code="NoSuchService"
968       ref="uuid:97326-A726F2-9FE326A-8C34ED3" />
969   </lu:Status>
970 </shps:DeleteResponse>
971
```

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

### 973 3.7.4. Delete Processing Rules

- 974 • If for any reason an delete is not accepted, the existing service instance at the SHPS **MUST NOT** be affected.
- 975 • Service instances created by one Advanced Client **MUST NOT** be visible to other Advanced Clients. If an  
976 Advanced Client presents a Service Handle for a service instance that the Advanced Client did not create, the  
977 SHPS **MUST** behave as if that service instance did not exist and accordingly any attempt to delete such a service  
978 should fail with an optional error code of *"NotFound."*
- 979 • Each `<shps:ServiceHandle>` is processed independently and may independently succeed or fail on its own  
980 merits.
- 981 • If all of the requested service handles are deleted, the top-level status code **MUST** be *"OK."* If all of the deletes  
982 failed, the top-level status code **MUST** be *"Failed."* Otherwise, if the results were mixed, the top-level status **MUST**  
983 be *"Partial"* and the second level status **MUST** be included for items for which the processing was not successful  
984 indicating so and including the `ref` attribute containing the `ServiceHandle` value for the delete(s) that failed.  
985 These second-level status codes **MAY** simply be *"Failed,"* or they may indicate with more detail the reason for the  
986 failure.
- 987 • If the top-level status code is *"Failed,"* the response **MAY** also contain other status codes (such as *Forbidden*) as a  
988 second-level status code. The SHPS may not wish to reveal the reason for failure, in which case no second-level  
989 status code will appear.

## 990 **3.8. Operation: *Invoke***

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

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

### 995 **3.8.1. wsa:Action values for Invoke Messages**

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

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

### 999 **3.8.2. Invoke Message**

1000 The <Invoke> request is called to invoke a service interface for a service instance hosted by the SHPS.

1001 The <shps:Invoke> request contains one or more <shps:InvokeItem> elements each of which have the  
1002 following attributes and/or elements:

1003 • **itemID: [Required]** - The itemID is a string that may have any value assigned by the requester (but MUST be  
1004 different than the itemID assigned to any other <shps:InvokeItem> elements in the same request.

1005 The itemID is used to enable correlation of the results in the <shps:InvokeResponse> with the  
1006 <shps:InvokeItem> that lead to those results.

1007 • <shps:ServiceHandle> **[Required]** - The Service Handle for the service instance which this invocation item  
1008 should apply to.

1009 • <xs:any> **[Required]** - The service level request (any request is allowed by the service instance exposed by  
1010 the SHPS). For example, if the Liberty People Service is the service that is being hosted by the SHPS, an  
1011 <ps:AddEntityRequest> may be specified in this location.

1012 The individual service definitions (such as the Liberty People Service Specification) drive what is allowable within  
1013 this location. Essentially, anything that MAY be placed into the body of a typical invocation of the service MAY  
1014 be placed into the <shps:InvokeItem> element.

1015 • **anyAttribute [Optional]** - Zero or more attributes from a namespace other than that of this specification. One  
1016 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

1017 The schema for the `<shps:Invoke>` is shown below.

```

1018 <!-- Invoke - invoke service level interface on hosted service instance -->
1019
1020 <xs:element name="Invoke" type="InvokeType"/>
1021
1022 <xs:complexType name="InvokeType">
1023   <xs:complexContent>
1024     <xs:extension base="RequestAbstractType">
1025       <xs:sequence>
1026         <xs:element ref="InvokeItem" maxOccurs="unbounded" />
1027       </xs:sequence>
1028     </xs:extension>
1029   </xs:complexContent>
1030 </xs:complexType>
1031
1032 <!-- InvokeItem - container for each service level request -->
1033
1034 <xs:element name="InvokeItem" type="InvokeItemType"/>
1035
1036 <xs:complexType name="InvokeItemType">
1037   <xs:sequence>
1038     <xs:element ref="ServiceHandle" />
1039     <xs:any namespace="##other"
1040       processContents="lax"
1041       minOccurs="0"
1042       maxOccurs="unbounded"/>
1043   </xs:sequence>
1044   <xs:attribute name="itemID" type="xs:string" use="required" />
1045   <xs:anyAttribute namespace="##other" processContents="lax"/>
1046 </xs:complexType>
1047

```

1048 **Figure 15. `<shps:Invoke>` — Schema Fragment**

1049 An example message body containing a `<Invoke>` message follows. This request invokes the Liberty Personal  
 1050 Profile `<pp:Modify>` interface on the instance of the service hosted by the SHPS (and referenced by the specified  
 1051 `<shps:ServiceHandle>`).

```

1052 <shps:Invoke>
1053   <shps:InvokeItem itemID="1">
1054     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1055     <pp:Modify>
1056       <pp:ModifyItem>
1057         ... modification data goes here ...
1058       </pp:ModifyItem>
1059     </pp:Modify>
1060   </shps:InvokeItem>
1061 </shps:Invoke>
1062

```

1063 **Example 19. Example `<shps:Invoke>` Message**

### 1064 3.8.3. InvokeResponse Message

1065 This response to the `<shps:Invoke>` request contains the following elements:

- 1066 • `<lu:Status>`: **[Required]** - The status of the response. See the processing rules below for more information.
- 1067 • `<shps:InvokeResponseItem>`: **[Optional]** - The container element for the response data for an  
 1068 `<shps:InvokeItem>` which contains the following attributes/elements:

- 1069     • **ref [Required]** The value from the `itemID` attribute on the `<shps:InvokeItem>` element in the request  
 1070     whose results are included in this `<shps:InvokeResponseItem>` element.
- 1071     • **<xs:any> [Required]** - The service level response for the request. For example, if the Liberty People Service  
 1072     `<ps:AddEntityRequest>` had been specified in the request, the `<ps:AddEntityResponse>` would be  
 1073     placed here.
- 1074     The individual service definitions (such as the Liberty People Service Specification) drive what is expected  
 1075     within this location. Essentially, anything that **MAY** be placed into the body of a typical response of the  
 1076     service **MAY** be placed into the `<shps:InvokeResponseItem>` element.
- 1077     • **anyAttribute [Optional]** - Zero or more attributes from a namespace other than that of this specification. One  
 1078     such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

```

1079 <!-- InvokeResponse - response to an Invoke request -->
1080 <xs:element name="InvokeResponse" type="InvokeResponseType"/>
1081
1082 <xs:complexType name="InvokeResponseType">
1083   <xs:complexContent>
1084     <xs:extension base="ResponseAbstractType">
1085       <xs:sequence>
1086         <xs:element ref="InvokeResponseItem" maxOccurs="unbounded" />
1087       </xs:sequence>
1088     </xs:extension>
1089   </xs:complexContent>
1090 </xs:complexType>
1091
1092 <!-- InvokeResponseItem - container for the result of each InvokeItem -->
1093 <xs:element name="InvokeResponseItem" type="InvokeResponseItemType"/>
1094
1095 <xs:complexType name="InvokeResponseItemType">
1096   <xs:sequence>
1097     <xs:any namespace="##other"
1098       processContents="lax"
1099       minOccurs="0"
1100       maxOccurs="unbounded"/>
1101   </xs:sequence>
1102   <xs:attribute name="ref" type="xs:string" use="required" />
1103 </xs:complexType>
1104
1105 </xs:complexType>
1106

```

1107                                   **Figure 16. <shps:InvokeResponse> — Schema Fragment**

1108   An example message body containing a `<InvokeResponse>` message follows. This is a successful response for a  
 1109   Liberty Personal Profile modification request.

```

1110 <shps:InvokeResponse>
1111   <lu:Status code="OK" />
1112   <shps:InvokeResponseItem ref="1">
1113     <pp:ModifyResponse>
1114       ... modification response data goes here ...
1115     </pp:ModifyResponse>
1116   </shps:InvokeResponseItem>
1117 </shps:InvokeResponse>
1118

```

1119                                   **Example 20. Example <shps:InvokeResponse> Message**

### 1120 3.8.4. Invoke Processing Rules

- 1121 • The CSI MAY invoke any service interface necessary to populate the SHPS hosted service instance with sufficient  
1122 data to properly respond to requests from WSCs.
- 1123 • The SHPS MAY process the requested invocation items in any order it sees fit. CSIs SHOULD NOT include  
1124 invocations of a service instance that are dependent upon each other as the results are not defined (the second  
1125 invocation may take place prior to the first).
- 1126 • Each `<shps:InvokeResponse>` element in the request MUST be treated independently.
- 1127 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code  
1128 MUST be "Failed"
- 1129 • If the top-level status code is "Failed," the response MAY also contain *Forbidden* as a second-level status code.  
1130 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will  
1131 appear.

### 1132 3.9. Operation: *GetStatus*

1133 The *GetStatus* operation is used by the Advanced Client to determine the status of a registered service instance at the  
1134 SHPS.

#### 1135 3.9.1. `wsa:Action` values for *GetStatus* Messages

1136 `<GetStatus>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2007-09:GetStatus"

1137 `<GetStatusResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of  
1138 "urn:liberty:shps:2007-09:GetStatusResponse."

#### 1139 3.9.2. *GetStatus* Message

1140 The `<GetStatus>` request is called to get the current status of the specified service instance at the SHPS.

1141 The `<shps:GetStatus>` request contains the following attributes and/or elements:

- 1142 • `<shps:ServiceHandle>` [Optional] Zero or more Service Handles for the service instances who's status is  
1143 requested. If not specified, the status of all service instances registered by the calling Advanced Client should be  
1144 returned.
- 1145 • `anyAttribute` [Optional] Zero or more attributes from a namespace other than that of this specification. One  
1146 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

1147 The schema for the `<shps:GetStatus>` is shown below.

```
1148 <!-- GetStatus - Get the status of a Service Instance at SHPS -->
1149
1150 <xs:element name="GetStatus" type="GetStatusType"/>
1151
1152 <xs:complexType name="GetStatusType">
1153   <xs:complexContent>
1154     <xs:extension base="RequestAbstractType">
1155       <xs:sequence>
1156         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1157       </xs:sequence>
1158     </xs:extension>
1159   </xs:complexContent>
1160 </xs:complexType>
1161
```

1162 **Figure 17. `<shps:GetStatus>` — Schema Fragment**

1163 An example message body containing a `<GetStatus>` message follows.

```
1164 <shps:GetStatus>
1165   <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1166   <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
1167 </shps:GetStatus>
1168
```

1169 **Example 21. Example `<shps:GetStatus>` Message**

### 1170 **3.9.3. GetStatusResponse Message**

1171 This response to the `<shps:GetStatus>` request contains the following elements and/or attributes:

- 1172 • `<lu:Status>`: **[Required]** - the status of the response. See the processing rules below for more information.
- 1173 • `<shps:GetStatusResponseItem>` : **[Optional]** - zero or more Service Handle/Service Status pairs indicating  
1174 the current status of the specified service. This element **MUST** be absent if there were no service instances  
1175 registered at the SHPS that met the conditions of the query or if the call otherwise failed.
- 1176 • `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One  
1177 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.



```

1178 <!-- GetStatusResponse - response to the GetStatus request -->
1179
1180 <xs:element name="GetStatusResponse" type="GetStatusResponseType"/>
1181
1182 <xs:complexType name="GetStatusResponseType">
1183   <xs:complexContent>
1184     <xs:extension base="ResponseAbstractType">
1185       <xs:sequence>
1186         <xs:element ref="GetStatusResponseItem" minOccurs="0"
1187           maxOccurs="unbounded" />
1188       </xs:sequence>
1189     </xs:extension>
1190   </xs:complexContent>
1191 </xs:complexType>
1192
1193 <!-- GetStatusResponseItem - the status of a single service instance -->
1194
1195 <xs:element name="GetStatusResponseItem" type="GetStatusResponseItemType"/>
1196
1197 <xs:complexType name="GetStatusResponseItemType">
1198   <xs:sequence>
1199     <xs:element ref="ServiceHandle" />
1200     <xs:element ref="ServiceStatus" />
1201   </xs:sequence>
1202   <xs:anyAttribute namespace="##other" processContents="lax"/>
1203 </xs:complexType>
1204
1205

```

1206 **Figure 18. <shps: GetStatusResponse> — Schema Fragment**

1207 An example message body containing a <GetStatusResponse> message follows. This is a successful response  
 1208 showing the status of two service instances.

```

1209 <shps: GetStatusResponse>
1210   <lu: Status code="OK" />
1211   <shps: GetStatusResponseItem>
1212     <shps: ServiceHandle>uuid:23023-023802-2032023-0238023</shps: ServiceHandle>
1213     <shps: ServiceStatus>urn:liberty:shps:2007-09:status:enabled</shps: ServiceStatus>
1214   </shps: GetStatusResponseItem>
1215   <shps: GetStatusResponseItem>
1216     <shps: ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps: ServiceHandle>
1217     <shps: ServiceStatus>urn:liberty:shps:2007-09:status:enabled</shps: ServiceStatus>
1218   </shps: GetStatusResponseItem>
1219 </shps: GetStatusResponse>
1220

```

1221 **Example 22. Example <shps: GetStatusResponse> Message**

### 1222 3.9.4. GetStatus Processing Rules

- 1223 • If there is no <ServiceHandle> on the request, the SHPS MUST interpret this request as a request for the status  
 1224 of ALL services instances registered at the SHPS by the invoking Advanced Client.

1225 • A request for a service instance which is NOT registered for the Advanced Client (whether or not it is reg-  
1226 istered for another Advanced Client) is, for the SHPS, considered successful and simply does not result in a  
1227 <GetStatusResponseItem> in the results of the operation. For example, if the <GetStatus> included a  
1228 single <ServiceHandle> referring to a service instance not currently registered at the SHPS, the response would  
1229 be:

```
1230 <shps:GetStatusResponse>  
1231 <!u:Status code="OK" />  
1232 </shps:GetStatusResponse>  
1233
```

1234 • If there are any matching results (even if there are some other non-matching elements), the request is successful  
1235 and the top-level status code MUST be *OK*. Otherwise, the top-level status code MUST be *Failed*.

1236 • If the top-level status code is "*Failed*," the response MAY also contain *NoResults* as a second-level status code.  
1237 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will  
1238 appear.

### 1239 3.10. Operation: *SetStatus*

1240 The *SetStatus* operation is used by the Advanced Client to enable or disable a registered service instance at the SHPS.

1241 This interface is typically used by the Advanced Client after it has completed the initialization of the hosted/proxied  
1242 service to make the service visible to WSCs.

1243 The primary effect of this call is that the SHPS registers and associates (or disassociates) the hosted or proxied service  
1244 instance in the principal's DS resource. This makes the service instance visible (or invisible) to WSCs.

#### 1245 3.10.1. *wsa:Action* values for *SetStatus* Messages

1246 <SetStatus> messages MUST include a <wsa:Action> SOAP header with the value of "urn:liberty:shps:2007-09:SetStatus"

1247 <SetStatusResponse> messages MUST include a <wsa:Action> SOAP header with the value of  
1248 "urn:liberty:shps:2007-09:SetStatusResponse."

#### 1249 3.10.2. *SetStatus* Message

1250 The <SetStatus> request is called to set the current status of the specified service instance to the specified status.

1251 The <shps:SetStatus> request contains the following attributes and/or elements:

1252 • *SetStatusItem*: **[Required]** - one or more set status items each of which contain:

1253 • *ServiceStatus*: **[Required]** - the new status for the specified service instance(s). This MUST be set to one  
1254 of the following values:

1255 • *urn:liberty:shps:2007-09:status:enabled*

1256 The service instance is to be made visible in the principal's DS resource such that WSCs can discover and  
1257 invoke it.

1258 • *urn:liberty:shps:2007-09:status:disabled*

1259 The service instance is to be removed from the principal's DS resource such that WSCs can no longer  
1260 discover and invoke it.

1261 • <shps:ServiceHandle> **[Required]** - one or more Service Handles for the service instances which are to  
1262 be enabled or disabled.

- 1263 • `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One  
1264 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

1265 The schema for the `<shps:SetStatus>` is shown below.

```
1266 <!-- SetStatus - set the status of a hosted/proxied service instance -->
1267
1268 <xs:element name="SetStatus" type="SetStatusType"/>
1269
1270 <xs:complexType name="SetStatusType">
1271   <xs:complexContent>
1272     <xs:extension base="RequestAbstractType">
1273       <xs:sequence>
1274         <xs:element ref="SetStatusItem" maxOccurs="unbounded" />
1275       </xs:sequence>
1276     </xs:extension>
1277   </xs:complexContent>
1278 </xs:complexType>
1279
1280 <xs:element name="SetStatusItem" type="SetStatusItemType"/>
1281
1282 <xs:complexType name="SetStatusItemType">
1283   <xs:sequence>
1284     <xs:element ref="ServiceStatus" />
1285     <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1286   </xs:sequence>
1287   <xs:attribute name="itemID" type="xs:string" use="required" />
1288 </xs:complexType>
1289
```

1290 **Figure 19. `<shps:SetStatus>` — Schema Fragment**

1291 An example message body containing a `<SetStatus>` message follows. This request enables two service instances.

```
1292 <shps:SetStatus>
1293   <shps:SetStatusItem itemID="1" >
1294     <shps:ServiceStatus>urn:liberty:shps:2007-09:status:disabled</shps:ServiceStatus>
1295     <shps:ServiceHandle>uuid:53723-123872-5223223-8237823</shps:ServiceHandle>
1296   </shps:SetStatusItem>
1297   <shps:SetStatusItem itemID="2">
1298     <shps:ServiceStatus>urn:liberty:shps:2007-09:status:enabled</shps:ServiceStatus>
1299     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1300     <shps:ServiceHandle>uuid:97326-A726F2-9FE326A-8C34ED3</shps:ServiceHandle>
1301   </shps:SetStatusItem>
1302 </shps:SetStatus>
1303
```

1304 **Example 23. Example `<shps:SetStatus>` Message**

### 1305 **3.10.3. SetStatusResponse Message**

1306 This response to the `<shps:SetStatus>` request contains the following elements:

- 1307 • `<lu:Status>`: **[Required]** - The status of the response. See the processing rules below for more information.
- 1308 • `anyAttribute` **[Optional]** - Zero or more attributes from a namespace other than that of this specification. One  
1309 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

```
1310 <!-- SetStatusResponse - response to the SetStatus request -->
1311
1312 <xs:element name="SetStatusResponse" type="SetStatusResponseType"/>
1313
1314 <xs:complexType name="SetStatusResponseType">
1315   <xs:complexContent>
1316     <xs:extension base="ResponseAbstractType" />
1317   </xs:complexContent>
1318 </xs:complexType>
1319
```

1320 **Figure 20. <shps:SetStatusResponse> — Schema Fragment**

1321 An example message body containing a <SetStatusResponse> message follows. This is a successful response.

```
1322 <shps:SetStatusResponse>
1323   <lu:Status code="OK" />
1324 </shps:SetStatusResponse>
1325
```

1326 **Example 24. Example <shps:SetStatusResponse> Message**

### 1327 3.10.4. SetStatus Processing Rules

- 1328 • This operation **MUST** be atomic and if successful, all portions of the request **MUST** have succeeded. If any  
1329 portion of the request fails, the entire request must fail. This requirement is mostly applicable in the case where  
1330 the request includes multiple Service Descriptors.
- 1331 • Each <shps:SetStatusItem> **MUST** be processed independently (even if one fails, the other  
1332 <shps:SetStatusItem> elements **MUST** be processed.
- 1333 • If request processing succeeded, the top-level status code **MUST** be "OK." Otherwise, the top-level status code  
1334 **MUST** be "Failed" If all <shps:SetStatusItem> requests are successfully processed, the top-level status  
1335 code **MUST** be "OK." If all of the items failed, the top-level status code **MUST** be "Failed." Otherwise, if the  
1336 results were mixed, the top-level status **MUST** be "Partial" and the second level status **MUST** be included for  
1337 items for which the processing was not successful indicating so and including the `ref` attribute containing the  
1338 `itemID` value for the item. These second-level status codes **MAY** simply be "Failed," or they may indicate with  
1339 more detail the reason for the failure.
- 1340 • If the top-level status code is "Failed," the response **MAY** also contain *NotFound* as a second-level status code.  
1341 The SHPS instance may not wish to reveal the reason for failure, in which case no second-level status code will  
1342 appear.

### 1343 **3.11. Operation: *Poll***

1344 The *Poll* operation is used by the CSI to poll for any new requests when the CSI is unable to expose an externally  
1345 visible endpoint for an incoming `shps:ProxyInvoke` interface for direct access by the SHPS.

1346 This operation is an adaptation of the *Poll* design pattern and inherits all of its structure and processing rules.

#### 1347 **3.11.1. `wsa:Action` values for `Poll` Messages**

1348 `<Poll>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:shps:2007-09:Poll."

1349 `<PollResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of  
1350 "urn:liberty:shps:2007-09:PollResponse."

#### 1351 **3.11.2. `Poll` Message**

1352 The `<Poll>` request is called by the CSI to ask the SHPS for any queued `shps:ProxyInvoke` requests and to  
1353 return any `shps:ProxyInvokeResponse` responses to prior requests.

1354 The structure of the `<shps:Poll>` request is derived from the structure of the `<dp:PollType>` without  
1355 modification. See [LibertyDP] for a complete description of the structure and meaning of the elements.

1356 The schema for the `<shps:Poll>` is shown below.

```
1357 <!-- Poll - Poll for new service requests -->  
1358  
1359 <xs:element name="Poll" type="dp:PollType"/>  
1360
```

1361 **Figure 21. `<shps:Poll>` — Schema Fragment**

1362 An example message body containing a `<shps:Poll>` message follows. This is a request asking for  
1363 `shps:ProxyInvoke` requests and asks SHPS to wait for 5 minutes if none are immediately available.

```
1364 <shps:Poll wait="300">  
1365 <wsa:Action>urn:liberty:shps:2007-09:ProxyInvoke</wsa:Action>  
1366 </shps:Poll>  
1367
```

1368 **Example 25. Example `<shps:Poll>` Message**

1369 Another example message body containing a `<shps:Poll>` message follows. This example also includes a  
1370 `shps:ProxyInvokeResponse` from a prior request received at the CSI.

```

1371 <shps:Poll wait="300">
1372   <wsa:Action>urn:liberty:shps:2007-09:ProxyInvoke</wsa:Action>
1373   <dp:Response ref="1">
1374     <shps:ProxyInvokeResponse>
1375       <lu:Status code="OK" />
1376       <shps:ProxyInvokeResponseItem ref="1">
1377         <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1378         <shps:ResponseHeaders>
1379           <sb:UsageDirectives> . . . . . </sb:UsageDirectives>
1380         </shps:ResponseHeaders>
1381         <pp:QueryResponse>
1382           . . . modification response data goes here . . .
1383         </pp:QueryResponse>
1384       </shps:ProxyInvokeResponseItem>
1385     </shps:ProxyInvokeResponse>
1386   </dp:Response>
1387 </shps:Poll>
1388
  
```

1389 **Example 26. Example `<shps:Poll>` Message with a `shps:ProxyInvokeResponse`.**

### 1390 3.11.3. PollResponse Message

1391 This response to the `<shps:Poll>` request is derived from the `<dp:PollResponseType>` without modification.  
 1392 See [\[LibertyDP\]](#) for a complete description of the structure and meaning of the elements.

```

1393 <!--PollResponse - response for the Poll request -->
1394
1395 <xs:element name="PollResponse" type="dp:PollResponseType"/>
1396
  
```

1397 **Figure 22. `<shps:PollResponse>` — Schema Fragment**

1398 An example message body containing a `<shps:PollResponse>` message follows. This is a successful response  
 1399 without an embedded `shps:ProxyInvoke` request (and therefore there were no queued requests) and SHPS is  
 1400 advising the CSI to poll again in 10 minutes (600 seconds).

```

1401 <shps:PollResponse nextPoll="600">
1402   <lu:Status code="OK" />
1403 </shps:PollResponse>
1404
  
```

1405 **Example 27. Example `<shps:PollResponse>` Message**

1406 Another example message body containing a `<shps:PollResponse>` message follows. This is a successful  
 1407 response with an embedded `shps:ProxyInvoke` request.

```

1408 <shps:PollResponse>
1409   <lu:Status code="OK" />
1410   <dp:Request itemID="1">
1411     <shps:ProxyInvoke>
1412       <shps:ProxyInvokeItem itemID="1">
1413         <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1414         <shps:InvocationContext>
1415           <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
1416           <shps:InvokingPrincipal
1417             Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
1418             NameQualifier="http://idps-r-us.com" >
1419             uuid:23923-023843-230932-230923
1420           </shps:InvokingPrincipal>
1421           <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
1422         </shps:InvocationContext>
1423         <shps:RequestHeaders>
1424           <sb:ProcessingContext>
1425             urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
1426           </sb:ProcessingContext>
1427         </shps:RequestHeaders>
1428         <pp:Query>
1429           ... query data goes here ...
1430         </pp:Query>
1431       </shps:ProxyInvokeItem>
1432     </shps:ProxyInvoke>
1433   </dp:Request>
1434 </shps:PollResponse>
1435

```

1436 **Example 28. Example <shps:PollResponse> Message**

### 1437 3.11.4. Poll Processing Rules

1438 • All of the processing rules defined for the *Poll* design pattern MUST be followed. See [LibertyDP] for further  
 1439 information.

## 1440 3.12. Operation: *ProxyInvoke*

1441 The *ProxyInvoke* operation is NOT exposed by the SHPS service, but rather MUST be exposed by an Advanced Client  
 1442 that uses the proxying capability of SHPS. This interface is what SHPS calls when it has been invoked in order to get  
 1443 the data to use in the response.

1444 The Advanced Client must either expose this interface directly via an externally visible endpoint (specified in the  
 1445 <shps:CallbackEPR> – see Section 3.2.6) or MUST use the <shps:Poll> interface to poll for invocations.

### 1446 3.12.1. wsa:Action values for ProxyInvoke Messages

1447 <ProxyInvoke> messages MUST include a <wsa:Action> SOAP header with the value of  
 1448 "urn:liberty:shps:2007-09:ProxyInvoke."

1449 <ProxyInvokeResponse> messages MUST include a <wsa:Action> SOAP header with the value of  
 1450 "urn:liberty:shps:2007-09:ProxyInvokeResponse."

### 1451 3.12.2. ProxyInvoke Message

1452 The <ProxyInvoke> request is called by the SHPS to pass on an invocation received from a WSC so that the  
 1453 Advanced Client can process the request and send the results to the SHPS who can then respond to the WSC.

1454 The <shps:ProxyInvoke> request contains one or more <shps:ProxyInvokeItem> elements each of which  
 1455 have the following attributes and/or elements:

- 1456 • `itemID`: **[Required]** - a string that may have any value assigned by the requester (but **MUST** be different than the  
1457 `itemID` assigned to any other `<shps:ProxyInvokeItem>` elements in the same request.
- 1458 The `itemID` is used to enable correlation of the results in the `<shps:ProxyInvokeResponse>` with the  
1459 `<shps:ProxyInvokeItem>` that lead to those results.
- 1460 • `<shps:ServiceHandle>` **[Required]** - the Service Handle for the service instance which this  
1461 `<shps:ProxyInvokeItem>` applies to. Different Service Handles **MAY** be specified in differ-  
1462 ent `<shps:ProxyInvokeItem>` elements in the same request if the Advanced Client used the same  
1463 `<shps:CallbackEPR>` for multiple service instances and multiple simultaneous invocations took place at the  
1464 SHPS.
- 1465 • `<shps:InvocationContext>` **[Required]** - the invocation context (see [Section 3.2.5](#) of the service instance at  
1466 the SHPS (necessary information for the CSI to make appropriate decisions about the results of the operation).
- 1467 • `<shps:RequestHeaders>` **[Optional]** - a container for zero or more SOAP headers from the call received by the  
1468 SHPS which the SHPS considers necessary for the CSI to resolve the request. Typical candidate headers include  
1469 (but are **not** limited to):
- 1470 • `<sb:ApplicationEPR>`
- 1471 • `<sb:Consent>`
- 1472 • `<sb:ProcessingContext>`
- 1473 • `<sb:Timeout>`
- 1474 • `<sb:UsageDirectives>`
- 1475 • `<xs:any>` **[Required]** - the unmodified incoming request that the SHPS received from the WSC. For example, if  
1476 the Liberty Profile Service is the service that is being proxied by the SHPS, an `<pp:Query>` may be specified  
1477 in this location.
- 1478 The individual service definitions (such as the Liberty Personal Profile Service Specification) drive what is  
1479 allowable within this location. Essentially, anything that **MAY** be placed into the body of a typical invocation of  
1480 the service **MAY** be placed into the `<shps:ProxyInvokeItem>` element.
- 1481 • `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One  
1482 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.



1483 The schema for the `<shps:ProxyInvoke>` is shown below.

```

1484 <!-- ProxyInvoke - proxied invocation of CSI -->
1485
1486 <xs:element name="ProxyInvoke" type="ProxyInvokeType"/>
1487
1488 <xs:complexType name="ProxyInvokeType">
1489   <xs:complexContent>
1490     <xs:extension base="RequestAbstractType">
1491       <xs:sequence>
1492         <xs:element ref="ProxyInvokeItem" maxOccurs="unbounded" />
1493       </xs:sequence>
1494     </xs:extension>
1495   </xs:complexContent>
1496 </xs:complexType>
1497
1498
1499 <!-- Declaration of ProxyInvokeItem element -->
1500 <xs:element name="ProxyInvokeItem" type="ProxyInvokeItemType"/>
1501
1502 <xs:complexType name="ProxyInvokeItemType">
1503   <xs:sequence>
1504     <xs:element ref="ServiceHandle" />
1505     <xs:element ref="InvocationContext" />
1506     <xs:element ref="RequestHeaders" minOccurs="0" />
1507     <xs:any namespace="##other"
1508       processContents="lax"
1509       minOccurs="0"
1510       maxOccurs="unbounded"/>
1511   </xs:sequence>
1512   <xs:attribute name="itemID" type="xs:string" use="required" />
1513   <xs:anyAttribute namespace="##other" processContents="lax"/>
1514 </xs:complexType>
1515
1516 <xs:element name="RequestHeaders" />
1517

```

1518 **Figure 23. `<shps:ProxyInvoke>` — Schema Fragment**

1519 An example message body containing a `<ProxyInvoke>` message follows. This is a proxied Liberty Personal Profile  
 1520 `<pp:Query>` request on the service instance referenced by the specified `<shps:ServiceHandle>`).

```

1521 <shps:ProxyInvoke>
1522   <shps:ProxyInvokeItem itemID="1">
1523     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1524     <shps:InvocationContext>
1525       <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
1526       <disco:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</disco:SecurityMechID>
1527     </shps:InvocationContext>
1528     <pp:Query>
1529       ... query data goes here ...
1530   </pp:Query>
1531 </shps:ProxyInvokeItem>
1532 </shps:ProxyInvoke>
1533

```

1534 **Example 29. Example `<shps:ProxyInvoke>` Message**

### 1535 **3.12.3. ProxyInvokeResponse Message**

1536 This response to the `<shps:ProxyInvoke>` request contains the following elements:

- 1537 • <lu:Status>: **[Required]** The status of the response. See the processing rules below for more information.
- 1538 • <shps:ProxyInvokeResponseItem> : **[Optional]** The container element for the response data for an
- 1539 <shps:ProxyInvokeItem> which contains the following attributes/elements:
  - 1540 • ref: **[Required]** - the value from the itemID attribute on the <shps:ProxyInvokeItem> element in the
  - 1541 request whose results are included in this <shps:ProxyInvokeResponseItem> element.
  - 1542 • <shps:ServiceHandle> **[Required]** - the Service Handle for the service instance which this
  - 1543 <shps:ProxyInvokeResponseItem> applies to.
  - 1544 • <shps:ResponseHeaders> **[Optional]** - a container for zero or more SOAP headers to be included on the
  - 1545 SHPS service response. Typical candidate headers include (but are **not** limited to):
    - 1546 • <sb:CredentialsContext>
    - 1547 • <sb:UsageDirectives>
    - 1548 • <sb:UserInteraction>
  - 1549 • <xs:any> **[Required]** - the service level response for the request. For example, if the Liberty People Service
  - 1550 <ps:AddEntityRequest> had been specified in the request, the <ps:AddEntityResponse> would be
  - 1551 placed here.
  - 1552 The individual service definitions (such as the Liberty People Service Specification) drive what is expected
  - 1553 within this location. Essentially, anything that MAY be placed into the body of a typical response of the
  - 1554 service MAY be placed into the <shps:ProxyInvokeResponseItem> element.
- 1555 • anyAttribute **[Optional]** - zero or more attributes from a namespace other than that of this specification. One
- 1556 such possibility is an **xs:ID** type attribute such as xml:id or wsu:Id.

```

1557 <!-- ProxyInvokeResponse - response to the ProxyInvoke request -->
1558
1559 <xs:element name="ProxyInvokeResponse" type="ProxyInvokeResponseType"/>
1560
1561 <xs:complexType name="ProxyInvokeResponseType">
1562   <xs:complexContent>
1563     <xs:extension base="ResponseAbstractType">
1564       <xs:sequence>
1565         <xs:element ref="ProxyInvokeResponseItem" minOccurs="0"
1566                       maxOccurs="unbounded" />
1567       </xs:sequence>
1568     </xs:extension>
1569   </xs:complexContent>
1570 </xs:complexType>
1571
1572 <!-- ProxyInvokeResponseItem - container for each service level invocation -->
1573
1574 <xs:element name="ProxyInvokeResponseItem" type="ProxyInvokeResponseItemType"/>
1575
1576 <xs:complexType name="ProxyInvokeResponseItemType">
1577   <xs:sequence>
1578     <xs:element ref="ServiceHandle" />
1579     <xs:element ref="ResponseHeaders" minOccurs="0" />
1580     <xs:any namespace="##other"
1581             processContents="lax"
1582             minOccurs="0"
1583             maxOccurs="unbounded"/>
1584   </xs:sequence>
1585   <xs:attribute name="ref" type="xs:string" use="required" />
1586 </xs:complexType>
1587
1588 <xs:element name="ResponseHeaders" />
1589
    
```

1590 **Figure 24. <shps:ProxyInvokeResponse> — Schema Fragment**

1591 An example message body containing a <ProxyInvokeResponse> message follows. This is a successful response  
1592 for a Liberty Personal Profile query request.

```
1593 <shps:ProxyInvokeResponse>
1594   <lu:Status code="OK" />
1595   <shps:ProxyInvokeResponseItem ref="1">
1596     <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
1597     <shps:ResponseHeaders>
1598       <sb:UsageDirectives> . . . . . </sb:UsageDirectives>
1599     </shps:ResponseHeaders>
1600     <pp:QueryResponse>
1601       . . . query response data goes here . . .
1602     </pp:QueryResponse>
1603   </shps:ProxyInvokeResponseItem>
1604 </shps:ProxyInvokeResponse>
1605
```

1606 **Example 30. Example <shps:ProxyInvokeResponse> Message**

1607 **3.12.4. ProxyInvoke Processing Rules**

- 1608 • The SHPS MUST build the <shps:InvocationContext> from the message it received from the WSC.
- 1609 • Each <shps:ProxyInvokeItem> MUST be processed independently (even if one fails, the other  
1610 <shps:ProxyInvokeItem> elements MUST be processed.
- 1611 • Each <shps:ProxyInvokeItem> MAY address different service instances by including different service  
1612 handles (if the CSI registered the same <shps:CallbackEPR> for multiple service instances.
- 1613 • If all <shps:ProxyInvokeItem> requests are successfully processed, the top-level status code MUST be  
1614 "OK." If all of the items failed, the top-level status code MUST be "Failed." Otherwise, if the results were mixed,  
1615 the top-level status MUST be "Partial" and the second level status MUST be included for items for which the  
1616 processing was not successful indicating so and including the ref attribute containing the itemID value for the  
1617 item. These second-level status codes MAY simply be "Failed," or they may indicate with more detail the reason  
1618 for the failure.
- 1619 Note that the status is of the processing of the <shps:ProxyInvokeItem> elements themselves, not the service  
1620 level processing (which would be reflected within the service response), So a <shps:ProxyInvokeResponse>  
1621 message MAY have a status code of "OK" while some service level responses may include status codes of "Failed"  
1622 (since the CSI successfully processed the ProxyRequest and is returning a failure service response).
- 1623 • If the top-level status code is "Failed," the response MAY also contain other status codes (such as *Forbidden*) as  
1624 a second-level status code. The SHPS instance may not wish to reveal the reason for failure, in which case no  
1625 second-level status code will appear.

## 1626 4. Service Hosting/Proxying Service Schema

```
1627 <?xml version="1.0" encoding="UTF-8"?>
1628 <xs:schema targetNamespace="urn:liberty:shps:2007-09"
1629     xmlns:lu="urn:liberty:util:2006-08"
1630     xmlns:disco="urn:liberty:disco:2006-08"
1631     xmlns:dp="urn:liberty:dp:2007-09"
1632     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1633     xmlns:wsa="http://www.w3.org/2005/08/addressing"
1634     xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
1635
1636     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
1637     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1638
1639     xmlns="urn:liberty:shps:2007-09"
1640     elementFormDefault="qualified"
1641     attributeFormDefault="unqualified"
1642 >
1643
1644 <xs:import namespace="urn:liberty:util:2006-08"
1645     schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1646
1647 <xs:import namespace="urn:liberty:disco:2006-08"
1648     schemaLocation="liberty-idwsf-disco-svc-v2.0.xsd"/>
1649
1650 <xs:import namespace="urn:liberty:dp:2007-09"
1651     schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>
1652
1653 <xs:import namespace="http://www.w3.org/2005/08/addressing"
1654     schemaLocation="http://www.w3.org/2005/08/addressing/ws-addr.xsd" />
1655
1656 <xs:import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
1657     schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd" />
1658
1659 <!-- Data Definitions { -->
1660
1661 <!-- ServiceHandle - a reference to a hosted/proxied service instance -->
1662
1663 <xs:element name="ServiceHandle" type="xs:anyURI"/>
1664
1665
1666 <!-- ServiceMode - the proxied or hosted mode of the service instance -->
1667
1668 <xs:element name="ServiceMode" type="xs:anyURI"/>
1669
1670
1671 <!-- ServiceStatus - the enabled/disabled status of the service instance -->
1672
1673 <xs:element name="ServiceStatus" type="xs:anyURI"/>
1674
1675
1676 <!-- InvocationContext - how the proxied service instance was invoked -->
1677
1678 <xs:element name="InvocationContext" type="InvocationContextType" />
1679
1680 <xs:complexType name="InvocationContextType">
1681     <xs:sequence>
1682         <xs:element ref="InvokingProvider" />
1683         <xs:element ref="InvokingPrincipal" minOccurs="0" />
1684         <xs:element ref="disco:SecurityMechID" />
1685     </xs:sequence>
1686     <xs:anyAttribute namespace="##other" processContents="lax"/>
1687 </xs:complexType>
1688
1689 <xs:element name="InvokingProvider" type="xs:string" />
1690 <xs:element name="InvokingPrincipal" type="saml2:NameIDType" />
1691
```

```
1692
1693 <!-- CallbackEPR - where the CSI can receive ProxyInvoke requests -->
1694
1695 <xs:element name="CallbackEPR" type="wsa:EndpointReferenceType"/>
1696
1697
1698
1699 <!-- End of Data Definitions } -->
1700
1701 <!-- Interface Definitions { -->
1702
1703
1704
1705 <!-- RequestAbstractType - common request message structure -->
1706
1707 <xs:complexType name="RequestAbstractType" abstract="true">
1708   <xs:anyAttribute namespace="##other" processContents="lax"/>
1709 </xs:complexType>
1710
1711
1712 <!-- ResponseAbstractType - common message response structure -->
1713
1714 <xs:complexType name="ResponseAbstractType" abstract="true">
1715   <xs:sequence>
1716     <xs:element ref="lu:Status"/>
1717   </xs:sequence>
1718   <xs:anyAttribute namespace="##other" processContents="lax"/>
1719 </xs:complexType>
1720
1721
1722 <!-- Delete - delete (remove) a service instance -->
1723
1724 <xs:element name="Delete" type="DeleteType"/>
1725
1726 <xs:complexType name="DeleteType">
1727   <xs:complexContent>
1728     <xs:extension base="RequestAbstractType">
1729       <xs:sequence>
1730         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1731       </xs:sequence>
1732     </xs:extension>
1733   </xs:complexContent>
1734 </xs:complexType>
1735
1736
1737 <!-- DeleteResponse - response to a delete request -->
1738
1739 <xs:element name="DeleteResponse" type="DeleteResponseType"/>
1740
1741 <xs:complexType name="DeleteResponseType">
1742   <xs:complexContent>
1743     <xs:extension base="ResponseAbstractType" />
1744   </xs:complexContent>
1745 </xs:complexType>
1746
1747
1748 <!-- GetStatus - Get the status of a Service Instance at SHPS -->
1749
1750 <xs:element name="GetStatus" type="GetStatusType"/>
1751
1752 <xs:complexType name="GetStatusType">
1753   <xs:complexContent>
1754     <xs:extension base="RequestAbstractType">
1755       <xs:sequence>
1756         <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1757       </xs:sequence>
1758     </xs:extension>
```

```
1759     </xs:complexContent>
1760 </xs:complexType>
1761
1762
1763 <!-- GetStatusResponse - response to the GetStatus request -->
1764
1765 <xs:element name="GetStatusResponse" type="GetStatusResponseType"/>
1766
1767 <xs:complexType name="GetStatusResponseType">
1768   <xs:complexContent>
1769     <xs:extension base="ResponseAbstractType">
1770       <xs:sequence>
1771         <xs:element ref="GetStatusResponseItem" minOccurs="0"
1772           maxOccurs="unbounded" />
1773       </xs:sequence>
1774     </xs:extension>
1775   </xs:complexContent>
1776 </xs:complexType>
1777
1778 <!-- GetStatusResponseItem - the status of a single service instance -->
1779
1780 <xs:element name="GetStatusResponseItem" type="GetStatusResponseItemType"/>
1781
1782 <xs:complexType name="GetStatusResponseItemType">
1783   <xs:sequence>
1784     <xs:element ref="ServiceHandle" />
1785     <xs:element ref="ServiceStatus" />
1786   </xs:sequence>
1787   <xs:anyAttribute namespace="##other" processContents="lax"/>
1788 </xs:complexType>
1789
1790
1791
1792 <!-- Query - query for ability to host or proxy services -->
1793
1794 <xs:element name="Query" type="disco:QueryType"/>
1795
1796
1797 <!-- QueryResponse - response for the Query request -->
1798
1799 <xs:element name="QueryResponse" type="disco:QueryResponseType"/>
1800
1801
1802 <!-- Invoke - invoke service level interface on hosted service instance -->
1803
1804 <xs:element name="Invoke" type="InvokeType"/>
1805
1806 <xs:complexType name="InvokeType">
1807   <xs:complexContent>
1808     <xs:extension base="RequestAbstractType">
1809       <xs:sequence>
1810         <xs:element ref="InvokeItem" maxOccurs="unbounded" />
1811       </xs:sequence>
1812     </xs:extension>
1813   </xs:complexContent>
1814 </xs:complexType>
1815
1816 <!-- InvokeItem - container for each service level request -->
1817
1818 <xs:element name="InvokeItem" type="InvokeItemType"/>
1819
1820 <xs:complexType name="InvokeItemType">
1821   <xs:sequence>
1822     <xs:element ref="ServiceHandle" />
1823     <xs:any namespace="##other"
1824       processContents="lax"
1825       minOccurs="0" />

```

```
1826         maxOccurs="unbounded"/>
1827     </xs:sequence>
1828     <xs:attribute name="itemID" type="xs:string" use="required" />
1829     <xs:anyAttribute namespace="##other" processContents="lax" />
1830 </xs:complexType>
1831
1832
1833 <!-- InvokeResponse - response to an Invoke request -->
1834
1835 <xs:element name="InvokeResponse" type="InvokeResponseType"/>
1836
1837 <xs:complexType name="InvokeResponseType">
1838     <xs:complexContent>
1839         <xs:extension base="ResponseAbstractType">
1840             <xs:sequence>
1841                 <xs:element ref="InvokeResponseItem" maxOccurs="unbounded" />
1842             </xs:sequence>
1843         </xs:extension>
1844     </xs:complexContent>
1845 </xs:complexType>
1846
1847 <!-- InvokeResponseItem - container for the result of each InvokeItem -->
1848
1849 <xs:element name="InvokeResponseItem" type="InvokeResponseItemType"/>
1850
1851 <xs:complexType name="InvokeResponseItemType">
1852     <xs:sequence>
1853         <xs:any namespace="##other"
1854             processContents="lax"
1855             minOccurs="0"
1856             maxOccurs="unbounded" />
1857     </xs:sequence>
1858     <xs:attribute name="ref" type="xs:string" use="required" />
1859 </xs:complexType>
1860
1861
1862 <!-- QueryRegistered - query for the registered service instances -->
1863
1864 <xs:element name="QueryRegistered" type="QueryRegisteredType"/>
1865
1866 <xs:complexType name="QueryRegisteredType">
1867     <xs:complexContent>
1868         <xs:extension base="RequestAbstractType">
1869             <xs:sequence>
1870                 <xs:element ref="ServiceHandle" minOccurs="0" maxOccurs="unbounded" />
1871             </xs:sequence>
1872         </xs:extension>
1873     </xs:complexContent>
1874 </xs:complexType>
1875
1876
1877 <!-- QueryRegisteredResponse - response for QueryRegistered request -->
1878
1879 <xs:element name="QueryRegisteredResponse" type="disco:QueryResponseType"/>
1880
1881
1882 <!-- Register - request for a new hosted or proxied service instance -->
1883
1884 <xs:element name="Register" type="RegisterType"/>
1885
1886 <xs:complexType name="RegisterType">
1887     <xs:complexContent>
1888         <xs:extension base="RequestAbstractType">
1889             <xs:sequence>
1890                 <xs:element ref="wsa:EndpointReference" maxOccurs="unbounded" />
1891             </xs:sequence>
1892         </xs:extension>
```

```
1893     </xs:complexContent>
1894 </xs:complexType>
1895
1896
1897 <!-- RegisterResponse - response to the Register request -->
1898
1899 <xs:element name="RegisterResponse" type="RegisterResponseType"/>
1900
1901 <xs:complexType name="RegisterResponseType">
1902   <xs:complexContent>
1903     <xs:extension base="ResponseAbstractType">
1904       <xs:sequence>
1905         <xs:element ref="RegisterResponseItem" maxOccurs="unbounded" />
1906       </xs:sequence>
1907     </xs:extension>
1908   </xs:complexContent>
1909 </xs:complexType>
1910
1911 <xs:element name="RegisterResponseItem" type="RegisterResponseItemType" />
1912
1913 <xs:complexType name="RegisterResponseItemType">
1914   <xs:sequence>
1915     <xs:element ref="ServiceHandle" />
1916   </xs:sequence>
1917   <xs:attribute name="ref" type="xs:string" use="required" />
1918 </xs:complexType>
1919
1920
1921 <!-- SetStatus - set the status of a hosted/proxied service instance -->
1922
1923 <xs:element name="SetStatus" type="SetStatusType"/>
1924
1925 <xs:complexType name="SetStatusType">
1926   <xs:complexContent>
1927     <xs:extension base="RequestAbstractType">
1928       <xs:sequence>
1929         <xs:element ref="SetStatusItem" maxOccurs="unbounded" />
1930       </xs:sequence>
1931     </xs:extension>
1932   </xs:complexContent>
1933 </xs:complexType>
1934
1935 <xs:element name="SetStatusItem" type="SetStatusItemType"/>
1936
1937 <xs:complexType name="SetStatusItemType">
1938   <xs:sequence>
1939     <xs:element ref="ServiceStatus" />
1940     <xs:element ref="ServiceHandle" maxOccurs="unbounded" />
1941   </xs:sequence>
1942   <xs:attribute name="itemID" type="xs:string" use="required" />
1943 </xs:complexType>
1944
1945
1946 <!-- SetStatusResponse - response to the SetStatus request -->
1947
1948 <xs:element name="SetStatusResponse" type="SetStatusResponseType"/>
1949
1950 <xs:complexType name="SetStatusResponseType">
1951   <xs:complexContent>
1952     <xs:extension base="ResponseAbstractType" />
1953   </xs:complexContent>
1954 </xs:complexType>
1955
1956
1957 <!-- Update - update the configuration of the hosted/proxied service instance -->
1958
1959 <xs:element name="Update" type="UpdateType"/>
```



```
1960
1961 <xs:complexType name="UpdateType">
1962   <xs:complexContent>
1963     <xs:extension base="RequestAbstractType">
1964       <xs:sequence>
1965         <xs:element ref="UpdateItem" maxOccurs="unbounded" />
1966       </xs:sequence>
1967     </xs:extension>
1968   </xs:complexContent>
1969 </xs:complexType>
1970
1971 <xs:element name="UpdateItem" type="UpdateItemType" />
1972
1973 <xs:complexType name="UpdateItemType">
1974   <xs:sequence>
1975     <xs:element ref="ServiceHandle" />
1976     <xs:element ref="wsa:EndpointReference" />
1977   </xs:sequence>
1978   <xs:attribute name="itemID" type="xs:string" use="required" />
1979 </xs:complexType>
1980
1981
1982 <!-- UpdateResponse - the response to the Update request -->
1983
1984 <xs:element name="UpdateResponse" type="UpdateResponseType"/>
1985
1986 <xs:complexType name="UpdateResponseType">
1987   <xs:complexContent>
1988     <xs:extension base="ResponseAbstractType" />
1989   </xs:complexContent>
1990 </xs:complexType>
1991
1992
1993 <!-- Poll - Poll for new service requests -->
1994
1995 <xs:element name="Poll" type="dp:PollType"/>
1996
1997
1998 <!-- PollResponse - response for the Poll request -->
1999
2000 <xs:element name="PollResponse" type="dp:PollResponseType"/>
2001
2002
2003 <!-- ProxyInvoke - proxied invocation of CSI -->
2004
2005 <xs:element name="ProxyInvoke" type="ProxyInvokeType"/>
2006
2007 <xs:complexType name="ProxyInvokeType">
2008   <xs:complexContent>
2009     <xs:extension base="RequestAbstractType">
2010       <xs:sequence>
2011         <xs:element ref="ProxyInvokeItem" maxOccurs="unbounded" />
2012       </xs:sequence>
2013     </xs:extension>
2014   </xs:complexContent>
2015 </xs:complexType>
2016
2017
2018 <!-- Declaration of ProxyInvokeItem element -->
2019 <xs:element name="ProxyInvokeItem" type="ProxyInvokeItemType"/>
2020
2021 <xs:complexType name="ProxyInvokeItemType">
2022   <xs:sequence>
2023     <xs:element ref="ServiceHandle" />
2024     <xs:element ref="InvocationContext" />
2025     <xs:element ref="RequestHeaders" minOccurs="0" />
2026     <xs:any namespace="##other"
```

```
2027         processContents="lax"
2028         minOccurs="0"
2029         maxOccurs="unbounded"/>
2030     </xs:sequence>
2031     <xs:attribute name="itemID" type="xs:string" use="required" />
2032     <xs:anyAttribute namespace="##other" processContents="lax"/>
2033 </xs:complexType>
2034
2035 <xs:element name="RequestHeaders" />
2036
2037
2038 <!-- ProxyInvokeResponse - response to the ProxyInvoke request -->
2039
2040 <xs:element name="ProxyInvokeResponse" type="ProxyInvokeResponseType"/>
2041
2042 <xs:complexType name="ProxyInvokeResponseType">
2043     <xs:complexContent>
2044         <xs:extension base="ResponseAbstractType">
2045             <xs:sequence>
2046                 <xs:element ref="ProxyInvokeResponseItem" minOccurs="0"
2047                             maxOccurs="unbounded" />
2048             </xs:sequence>
2049         </xs:extension>
2050     </xs:complexContent>
2051 </xs:complexType>
2052
2053 <!-- ProxyInvokeResponseItem - container for each service level invocation -->
2054
2055 <xs:element name="ProxyInvokeResponseItem" type="ProxyInvokeResponseItemType"/>
2056
2057 <xs:complexType name="ProxyInvokeResponseItemType">
2058     <xs:sequence>
2059         <xs:element ref="ServiceHandle" />
2060         <xs:element ref="ResponseHeaders" minOccurs="0" />
2061         <xs:any namespace="##other"
2062                 processContents="lax"
2063                 minOccurs="0"
2064                 maxOccurs="unbounded"/>
2065     </xs:sequence>
2066     <xs:attribute name="ref" type="xs:string" use="required" />
2067 </xs:complexType>
2068
2069 <xs:element name="ResponseHeaders" />
2070
2071
2072
2073 <!-- End of Interface Definitions } -->
2074
2075 </xs:schema>
2076
```

## 2077 5. Service Hosting/Proxying Service WSDL

```
2078 <?xml version="1.0"?>
2079 <definitions name="shps-svc"
2080   targetNamespace="urn:liberty:shps:2007-09"
2081   xmlns:tns="urn:liberty:shps:2007-09"
2082   xmlns="http://schemas.xmlsoap.org/wsdl/"
2083   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2084   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2085   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2086   xmlns:shps="urn:liberty:shps:2007-09"
2087   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2088   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2089     http://schemas.xmlsoap.org/wsdl/
2090     http://www.w3.org/2006/02/addressing/wsdl
2091     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2092
2093   <types>
2094     <xsd:schema>
2095       <xsd:import namespace="urn:liberty:shps:2007-09"
2096         schemaLocation="liberty-idwsf-shps-v1.0.xsd" />
2097     </xsd:schema>
2098   </types>
2099
2100   <message name="Query">
2101     <part name="body" element="shps:Query"/>
2102   </message>
2103   <message name="QueryResponse">
2104     <part name="body" element="shps:QueryResponse"/>
2105   </message>
2106
2107   <message name="QueryRegistered">
2108     <part name="body" element="shps:QueryRegistered"/>
2109   </message>
2110   <message name="QueryRegisteredResponse">
2111     <part name="body" element="shps:QueryRegisteredResponse"/>
2112   </message>
2113
2114   <message name="Register">
2115     <part name="body" element="shps:Register"/>
2116   </message>
2117   <message name="RegisterResponse">
2118     <part name="body" element="shps:RegisterResponse"/>
2119   </message>
2120
2121   <message name="Update">
2122     <part name="body" element="shps:Update"/>
2123   </message>
2124   <message name="UpdateResponse">
2125     <part name="body" element="shps:UpdateResponse"/>
2126   </message>
2127
2128   <message name="Delete">
2129     <part name="body" element="shps>Delete"/>
2130   </message>
2131   <message name="DeleteResponse">
2132     <part name="body" element="shps>DeleteResponse"/>
2133   </message>
2134
2135   <message name="Invoke">
2136     <part name="body" element="shps:Invoke"/>
2137   </message>
2138   <message name="InvokeResponse">
2139     <part name="body" element="shps:InvokeResponse"/>
2140   </message>
2141
2142   <message name="GetStatus">
```

```
2143     <part name="body" element="shps:GetStatus" />
2144 </message>
2145 <message name="GetStatusResponse">
2146   <part name="body" element="shps:GetStatusResponse" />
2147 </message>
2148
2149 <message name="SetStatus">
2150   <part name="body" element="shps:SetStatus" />
2151 </message>
2152 <message name="SetStatusResponse">
2153   <part name="body" element="shps:SetStatusResponse" />
2154 </message>
2155
2156 <portType name="SHPSPort">
2157
2158   <operation name="Query">
2159     <input message="tns:Query"
2160       wsaw:Action="urn:liberty:shps:2007-09:Query" />
2161     <output message="tns:QueryResponse"
2162       wsaw:Action="urn:liberty:shps:2007-09:QueryResponse" />
2163   </operation>
2164
2165   <operation name="QueryRegistered">
2166     <input message="tns:QueryRegistered"
2167       wsaw:Action="urn:liberty:shps:2007-09:QueryRegistered" />
2168     <output message="tns:QueryRegisteredResponse"
2169       wsaw:Action="urn:liberty:shps:2007-09:QueryRegisteredResponse" />
2170   </operation>
2171
2172   <operation name="Register">
2173     <input message="tns:Register"
2174       wsaw:Action="urn:liberty:shps:2007-09:Register" />
2175     <output message="tns:RegisterResponse"
2176       wsaw:Action="urn:liberty:shps:2007-09:RegisterResponse" />
2177   </operation>
2178
2179   <operation name="Update">
2180     <input message="tns:Update"
2181       wsaw:Action="urn:liberty:shps:2007-09:Update" />
2182     <output message="tns:UpdateResponse"
2183       wsaw:Action="urn:liberty:shps:2007-09:UpdateResponse" />
2184   </operation>
2185
2186   <operation name="Delete">
2187     <input message="tns>Delete"
2188       wsaw:Action="urn:liberty:shps:2007-09>Delete" />
2189     <output message="tns>DeleteResponse"
2190       wsaw:Action="urn:liberty:shps:2007-09>DeleteResponse" />
2191   </operation>
2192
2193   <operation name="Invoke">
2194     <input message="tns:Invoke"
2195       wsaw:Action="urn:liberty:shps:2007-09:Invoke" />
2196     <output message="tns:InvokeResponse"
2197       wsaw:Action="urn:liberty:shps:2007-09:InvokeResponse" />
2198   </operation>
2199
2200   <operation name="GetStatus">
2201     <input message="tns:GetStatus"
2202       wsaw:Action="urn:liberty:shps:2007-09:GetStatus" />
2203     <output message="tns:GetStatusResponse"
2204       wsaw:Action="urn:liberty:shps:2007-09:GetStatusResponse" />
2205   </operation>
2206
2207   <operation name="SetStatus">
2208     <input message="tns:SetStatus"
2209       wsaw:Action="urn:liberty:shps:2007-09:SetStatus" />
```

```
2210     <output message="tns:SetStatusResponse"
2211         wsaw:Action="urn:liberty:shps:2007-09:SetStatusResponse" />
2212     </operation>
2213
2214 </portType>
2215
2216 <!--
2217 An example of a binding and service that can be used with this
2218 abstract service description is shpsided below.
2219 -->
2220
2221 <binding name="SHPSBinding" type="tns:SHPSPort">
2222
2223     <soap:binding style="document"
2224         transport="http://schemas.xmlsoap.org/soap/http"/>
2225
2226     <operation name="Query">
2227         <input> <soap:body use="literal"/> </input>
2228         <output> <soap:body use="literal"/> </output>
2229     </operation>
2230
2231     <operation name="QueryRegistered">
2232         <input> <soap:body use="literal"/> </input>
2233         <output> <soap:body use="literal"/> </output>
2234     </operation>
2235
2236     <operation name="Register">
2237         <input> <soap:body use="literal"/> </input>
2238         <output> <soap:body use="literal"/> </output>
2239     </operation>
2240
2241     <operation name="Update">
2242         <input> <soap:body use="literal"/> </input>
2243         <output> <soap:body use="literal"/> </output>
2244     </operation>
2245
2246     <operation name="Delete">
2247         <input> <soap:body use="literal"/> </input>
2248         <output> <soap:body use="literal"/> </output>
2249     </operation>
2250
2251     <operation name="Invoke">
2252         <input> <soap:body use="literal"/> </input>
2253         <output> <soap:body use="literal"/> </output>
2254     </operation>
2255
2256     <operation name="GetStatus">
2257         <input> <soap:body use="literal"/> </input>
2258         <output> <soap:body use="literal"/> </output>
2259     </operation>
2260
2261     <operation name="SetStatus">
2262         <input> <soap:body use="literal"/> </input>
2263         <output> <soap:body use="literal"/> </output>
2264     </operation>
2265
2266 </binding>
2267
2268 <service name="SHPSService">
2269
2270     <port name="SHPSPort" binding="tns:SHPSBinding">
2271
2272         <!-- Modify with the REAL SOAP endpoint -->
2273
2274         <soap:address location="http://example.com/shps"/>
2275
2276     </port>
```

```
2277
2278   </service>
2279
2280 </definitions>
2281
```

## 2282 References

### 2283 Normative

- 2284 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-  
2285 errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>
- 2286 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"  
2287 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 2288 [LibertyDP] Cahill, Conor P., eds. "Liberty ID-WSF Design Patterns Specification," Version 1.0, Liberty Alliance  
2289 Project (14 December 2007). <http://www.projectliberty.org/specs>
- 2290 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-  
2291 errata-v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>
- 2292 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version  
2293 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 2294 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14  
2295 December, 2004). <http://www.projectliberty.org/specs>
- 2296 [LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,  
2297 Single Sign-On, and Identity Mapping Services Specification," v2.0-errata-1.0-01, Liberty Alliance Project  
2298 (28 November, 2006). <http://www.projectliberty.org/specs>
- 2299 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty  
2300 ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007).  
2301 <http://www.projectliberty.org/specs>
- 2302 [LibertyPAOS] Aarts, Robert, Kemp, John, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version  
2303 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 2304 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet  
2305 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 2306 [RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251,  
2307 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2251.txt>
- 2308 [RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A.  
2309 Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force  
2310 <http://www.ietf.org/rfc/rfc2252.txt>
- 2311 [RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search  
2312 Results," RFC 2891, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2891.txt>
- 2313 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Pro-  
2314 tocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS  
2315 Standard, Organization for the Advancement of Structured Information Standards [http://www.oasis-  
open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf](http://www.oasis-<br/>2316 open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 2317 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions  
2318 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-  
2319 SIS Standard, Organization for the Advancement of Structured Information Standards [http://docs.oasis-  
open.org/security/saml/v2.0/saml-core-2.0-os.pdf](http://docs.oasis-<br/>2320 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)

- 2321 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,  
2322 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"  
2323 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards  
2324 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- 2325 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October  
2326 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium  
2327 <http://www.w3.org/TR/xmlschema-1/>
- 2328 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman,  
2329 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C  
2330 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 2331 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).  
2332 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium  
2333 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 2334 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and  
2335 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 2336 [XPath] Clark, J., DeRose, S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0,"  
2337 Recommendation, W3C <http://www.w3.org/TR/xpath> [August 2003].

## 2338 Informative

- 2339 [LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty  
2340 Alliance Project (13 January, 2005). <http://www.projectliberty.org/specs>
- 2341 [LibertyIDPPGuide] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Implementation  
2342 Guidelines," Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 2343 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework  
2344 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 2345 [LibertyACT] Cahill, Conor P., eds. "Liberty Advanced Client Technologies," Version 1.0, Liberty Alliance Project  
2346 (14 December 2007). <http://www.projectliberty.org/specs>