



# Liberty ID-WSF Design Patterns

Version: 1.0

**Editors:**

Conor P. Cahill, Intel Corporation

**Contributors:**

George Fletcher, AOL LLC

Hubert Le Van Gong, Sun

Paul Madsen, NTT

Hiroyoshi Takiguchi, NTT

Greg Whitehead, Hewlett-Packard

**Abstract:**

This specification defines common design patterns that can be included in other Liberty ID-WSF specifications.

**Filename:** liberty-idwsf-dp-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	<b>Contents</b>	
42	1. Introduction	4
43	1.1. Notation and Conventions	4
44	1.1.1. XML Namespaces	4
45	2. Polling	5
46	2.1. <code>PollType</code> Message Type	5
47	2.2. <code>PollResponseType</code> Message Type	6
48	2.3. Poll Processing Rules	8
49	3. Pagination	10
50	3.1. Basic Pagination	10
51	3.1.1. Basic Pagination - Request attributes	10
52	3.1.2. Basic Pagination - Response Attributes	11
53	3.2. Extended Pagination	11
54	3.2.1. Extended Pagination - Request attributes	12
55	3.2.2. Extended Pagination - Response Attributes	13
56	3.3. Pagination Examples	13
57	3.3.1. Pagination Schema Example	13
58	3.3.2. Basic Pagination Example	14
59	3.3.3. Extended Pagination Example	15
60	3.4. Pagination Processing Rules	17
61	4. Delayed Notification	18
62	4.1. Delayed Notification Sequence	18
63	4.1.1. Step 1: Invoker submits request with <code>&lt;dp:NotifyTo&gt;</code> element	18
64	4.1.2. Step 2: Service instance performs initial processing	18
65	4.1.3. Step 3: The operation completes	19
66	4.2. Delayed Notification data structures	19
67	4.2.1. <code>&lt;dp:NotifyTo&gt;</code> Element	19
68	4.3. Delayed Notification Operations	21
69	4.3.1. Operation: <i>Notification</i>	21
70	4.4. Delayed Notification Examples (Non-Normative)	23
71	4.4.1. Step 1: Single request to update several PMDs	23
72	4.4.2. Step 2: Initial response from the Provisioning Service	25
73	4.4.3. Step 3: First Notification	26
74	4.4.4. Step 4: Second Notification	26
75	4.4.5. Step 5: Final Notification	26
76	5. ID-WSF Design Patterns Schema	28
77	6. Notification Endpoint WSDL (non-normative)	31
78	References	33

## 79 **1. Introduction**

80 This specification defines common design patterns that can be included in other Liberty ID-WSF specifications.

### 81 **1.1. Notation and Conventions**

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

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

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

#### 89 **1.1.1. XML Namespaces**

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

91 • The prefix *dp*: represents the Design Patterns namespace. This namespace is the default for instance fragments,  
92 type names, and element names in this document. In schema listings, and in example messages and fragments  
93 thereof, this is the default namespace *when* no prefix is shown:

94 *urn:liberty:dp:2007-09*

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

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

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

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

## 99 2. Polling

100 The *Poll* operation is used when an entity can't expose an incoming request communications channel and therefore  
101 must poll its consumer(s) for requests.

### 102 2.1. PollType Message Type

103 The `<dp:PollType>` element defines a structure for a poll interface request which is called to return the results of a  
104 previous request and/or to ask for one or more new requests.

105 The `<dp:PollType>` contains the following attributes and/or elements:

106 • `<wsa:Action>` **[Optional]** - the Action URIs for the requests that the invoking party is willing to accept in the  
107 response. This is a hint to help the recipient of the `<dp:Poll>` decide which request(s) can be sent. If not  
108 specified the recipient can send whatever it wants.

109 • `<dp:Response>` **[Optional]** - a container for the response element(s) from previous request(s). Note that  
110 the number of response element(s) **MUST** match the number of request elements that were present in the  
111 `<dp:PollResponse>`, which resulted in this `<dp:Poll>`.

112 The `<dp:Response>` element has the following attributes/elements:

113 • `<xs:any>` **[Required]** - the service level response (any response is allowed. For example, if the invoking party  
114 hosted a Liberty People Service and had received a People Service request, the service level response could be  
115 a `<ps:AddEntityResponse>` element in this location.

116 The individual service definitions (such as the Liberty People Service Specification) drive what is allowable  
117 within this location. Essentially, anything that **MAY** be placed into the body of a typical response to an  
118 invocation of the service **MAY** be placed into the `<dp:Response>` element.

119 There **MUST ONLY** be a single service response within a given `<dp:Response>` element. If multiple  
120 service responses are to be included in a single `<dp:Poll>` message they **MUST** be placed into separate  
121 `<dp:Response>` elements.

122 • `ref` **[required]** - a reference to the `itemID` in the request for which this response was generated.

123 • `wait` **[Required]** - an attribute indicating the number of seconds the invoker wants the recipient to wait if there  
124 aren't any requests available immediately. If set to zero, no wait takes place. The recipient **MAY** limit the amount  
125 of time it is willing to wait for more requests.

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

128 The schema for the `<dp:PollType>` is shown below.

```

129 <!-- PollType - datatype for polling the recipient for any new work -->
130
131 <xs:complexType name="PollType">
132   <xs:sequence>
133     <xs:element ref="wsa:Action" minOccurs="0" maxOccurs="unbounded" />
134     <xs:element ref="Response" minOccurs="0" maxOccurs="unbounded" />
135   </xs:sequence>
136   <xs:attribute name="wait" type="xs:integer" use="required" />
137   <xs:anyAttribute namespace="##other" processContents="lax"/>
138 </xs:complexType>
139
140 <xs:element name="Response" type="ResponseType" />
141 <xs:complexType name="ResponseType">
142   <xs:sequence>
143     <xs:any namespace="##other"
144       processContents="lax"
145       minOccurs="0"
146       maxOccurs="unbounded" />
147   </xs:sequence>
148   <xs:attribute name="ref" type="xs:string" use="required" />
149 </xs:complexType>
150

```

151 **Figure 1. `<dp:PollType>` — Schema Fragment**

152 An example message body containing a message derived from the `<dp:PollType>` follows. This indicates that the  
153 caller is willing to wait 5 minutes for any new requests and is looking for a `<shps:ProxyInvoke>` request.

```

154 <shps:Poll wait="300">
155   <wsa:Action>urn:liberty:shps:2007-09:ProxyInvoke</wsa:Action>
156 </shps:Poll>
157

```

158 **Example 1. Example request derived from the `<dp:PollType>` datatype**

159 An example message body containing a message derived from the `<dp:PollType>` with a response from a prior  
160 request follows.

```

161 <shps:Poll wait="300">
162   <wsa:Action>urn:liberty:shps:2007-09:ProxyInvoke</wsa:Action>
163   <dp:Response ref="1">
164     <shps:ProxyInvokeResponse>
165       <lu:Status code="OK" />
166       <shps:ProxyInvokeResponseItem ref="1">
167         <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
168         <shps:ResponseHeaders>
169           <sb:UsageDirectives> . . . . . </sb:UsageDirectives>
170         </shps:ResponseHeaders>
171         <pp:QueryResponse>
172           . . . modification response data goes here . . .
173         </pp:QueryResponse>
174       </shps:ProxyInvokeResponseItem>
175     </shps:ProxyInvokeResponse>
176   </dp:Response>
177 </shps:Poll>
178

```

179 **Example 2. Example request derived from the `<dp:PollType>` datatype**

## 180 2.2. PollResponseType Message Type

181 This response type is used to define responses to messages derived from the `<dp:PollType>` message type and  
182 contains the following elements:

- 183 • `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- 184 • `<dp:Request>` **[Optional]** - zero or more request container(s) each of which contain the following  
185 attributes/elements:
  - 186 • `<xs:any>` **[Required]** - the service level request message. For example, if the Liberty People Service is the  
187 service hosted by the polling entity, a `<ps:AddEntityRequest>` may be specified in this location.

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

191 There **MUST ONLY** be a single service request within a given `<dp:Request>` element. If multiple service  
192 requests are to be included in a single `<dp:PollResponse>` message, they **MUST** be placed into separate  
193 `<dp:Request>` elements.
- 194 • `itemID` **[required]** - the identifier for this request (for correlation with the results in the response).
- 195 • `nextPoll` **[Optional]** - the number of seconds that the caller should wait before polling again for more requests.  
196 This attribute has no effect and **SHOULD NOT** be specified if a `<dp:Request>` is included in the message as the  
197 polling entity **SHOULD** poll again as soon as the response for that request is ready (and, of course, include the  
198 response in the poll).
- 199 • `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One  
200 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

```
201 <!-- PollResponseType - the datatype of response to a polling message -->
202
203 <xs:complexType name="PollResponseType">
204   <xs:complexContent>
205     <xs:extension base="ResponseAbstractType">
206       <xs:sequence>
207         <xs:element ref="Request" minOccurs="0" maxOccurs="unbounded" />
208       </xs:sequence>
209       <xs:attribute name="nextPoll" type="xs:integer" use="optional" />
210     </xs:extension>
211   </xs:complexContent>
212 </xs:complexType>
213
214 <xs:element name="Request" type="RequestType" />
215 <xs:complexType name="RequestType">
216   <xs:sequence>
217     <xs:any namespace="##other"
218       processContents="lax"
219       minOccurs="0"
220       maxOccurs="unbounded" />
221   </xs:sequence>
222   <xs:attribute name="itemID" type="xs:string" use="required" />
223 </xs:complexType>
224
```

225 **Figure 2.** `<dp:PollResponseType>` — Schema Fragment

226 An example message body containing a message derived from the `<dp:PollResponseType>` follows. This is a  
227 successful SHPS response with no new requests and a request for the caller to wait 10 minutes before polling again.

```
228 <shps:PollResponse nextPoll="600">
229   <lu:Status code="OK" />
230 </shps:PollResponse>
231
```

232 **Example 3. Example message derived from the `<dp:PollResponseType>` Message Type**

233 Another example message body containing a message derived from the `<dp:PollResponseType>` follows. This  
234 is a successful response message with a `<shps:ProxyInvoke>` request.

```
235 <shps:PollResponse>
236   <lu:Status code="OK" />
237   <dp:Request itemID="1">
238     <shps:ProxyInvoke>
239       <shps:ProxyInvokeItem itemID="1">
240         <shps:ServiceHandle>uuid:23023-023802-2032023-0238023</shps:ServiceHandle>
241         <shps:InvocationContext>
242           <shps:InvokingProvider>http://services.corp.com</shps:InvokingProvider>
243           <shps:InvokingPrincipal
244             Format="urn:oasis:tc:SAML:2.0:namid-format:persistent"
245             NameQualifier="http://idps-r-us.com" >
246             uuid:23923-023843-230932-230923
247           </shps:InvokingPrincipal>
248           <disco:SecurityMechID>urn:liberty:ds:2006-08:TLS:SAMLV2</disco:SecurityMechID>
249         </shps:InvocationContext>
250         <shps:RequestHeaders>
251           <sb:ProcessingContext>
252             urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
253           </sb:ProcessingContext>
254         </shps:RequestHeaders>
255         <pp:Query>
256           ... query data goes here ...
257         </pp:Query>
258       </shps:ProxyInvokeItem>
259     </shps:ProxyInvoke>
260   </dp:Request>
261 </shps:PollResponse>
262
```

263 **Example 4. Example message derived from the `<dp:PollResponseType>` Message Type**

## 264 2.3. Poll Processing Rules

- 265 • If the request includes one or more `<wsa:Action>` elements, the recipient **MUST NOT** include request  
266 elements in the response that are not associated with the specified action value(s).
- 267 • If the request includes the `wait` attribute, the recipient **SHOULD** respond immediately when it has requests for  
268 the invoker (including any that arrive after the recipient has started waiting for new requests). The recipient **MAY**  
269 use the specified amount of time on the `wait` to determine how long it is willing to wait before responding with  
270 no request.
- 271 • A single poll response **SHOULD NOT** include multiple requests which make use of Delayed Notification  
272 ([Section 4](#)) as this can result in indeterminate results when the results are subsequently returned in different  
273 notification using the same `ref` attribute value.
- 274 Requests making use of Delayed Notification **SHOULD** be sent in separate poll response messages.
- 275 • If the response is to be sent without an embedded request, the response should still be treated as a successful  
276 response. The lack of a `<dp:Request>` is sufficient to determine that there is no request and the poll operation  
277 itself was successful.

- 
- 278 • If the response includes a `nextPoll` attribute, the invoker SHOULD NOT poll the recipient again until that amount  
279 of time has passed.
- 280 The `nextPoll` SHOULD NOT be specified on responses that include an incoming request as the invoker should  
281 issue a new poll containing the response to the request as soon as it has completed processing the request. If the  
282 attribute is specified in such cases, the invoker MAY ignore it.
- 283 If a poll is received too soon following a response with a `nextPoll` attribute, the recipient MAY reject the request  
284 and in doing so, MAY indicate a secondary status code of "*PollTooSoon*."
- 285 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code  
286 MUST be "*Failed*".
- 287 • If the top-level status code is "*Failed*," the response MAY also contain *Forbidden* as a second-level status code.  
288 The SHP Service instance may not wish to reveal the reason for failure, in which case no second-level status code  
289 will appear.

## 290 3. Pagination

291 Pagination is an add-on functionality to an interface to allow the requester to page through results rather than forcing  
292 the entire set of results into a single response. This capability is very useful when a large set of results may be returned  
293 in response to a request.

294 Pagination also brings up the issue of transactional consistency. This is due to the fact that time passes between the  
295 request for the first page and the request for the second page and there may have been changes made to the data during  
296 that time (even if the time was very small).

297 For example, after reading the first page of 10 items, a new item is added that would be sorted into the results as item  
298 3 (and therefore would have been on the first page if it had been three at the time the first page was requested). With  
299 the subsequent request for the next 10 items, which items are returned – the original items 11-20, or the original items  
300 10-19 (which are now 11-20 because of the new item in position 3).

301 Transactional consistency is **not** always a problem. Many operations are either static in nature, (such as reading a  
302 static document like the one you're reading now), in other cases it just doesn't matter, especially considering that the  
303 likelihood of the collision is relatively small and the ramifications are minimal – such as listing web pages that contain  
304 a particular term.

305 Of course, there are situations where transactional consistency is important and so we need to support both situations.  
306 Therefore we have defined two attribute groups to support both situations:

307 • **Basic** - support for basic pagination with no explicit support for consistency control. This is typically used when  
308 the data is static or consistency just doesn't matter. If a caller is still concerned about consistence, they can just  
309 read the entire data set in a single operation.

310 • **Extended** - support for basic pagination and support for a snapshot type of consistency control (the server collects  
311 the data that would be included in the response at the start of the pagination of results and guarantees that only  
312 those records are included in subsequent pages through the result set).

313 In all cases, pagination refers to accessing a limited set of the *item* that may be in the result set. The interface that  
314 adopts this design pattern **MUST** define what is considered an *item*. In many cases, the *item* will be the native object  
315 or record that the request is attempting to access. In other cases, the *item* will equate to a byte of data in the response.

### 316 3.1. Basic Pagination

#### 317 3.1.1. Basic Pagination - Request attributes

318 Basic pagination provides for paging through results of an operation without any support for consistency level controls.  
319 The attribute group `dp:PaginationBasicAttributeGroup` contains the following attributes:

320 • `count` [**Optional**] - the maximum number of *items* to be included in the response to this request. The actual  
321 number returned **MAY** be smaller if the number of *items* remaining in the result set is less than the specified count  
322 or at the discretion of the service.

323 If this attribute is not specified, the request **MUST** be interpreted as if the entire remaining set of *items* are  
324 requested.

325 • `offset` [**Optional**] - the starting location in the result set for the *items* to be returned for this request. The first  
326 *item* has the offset zero.

327 If this attribute is not specified, the request **MUST** be interpreted as if zero had been specified.

328 The schema for the `dp:BasicPagingAttributeGroup` is shown below.

```
329 <!--BasicPagingAttributeGroup - basic request pagination support -->
330
331 <xs:attributeGroup name="BasicPagingAttributeGroup" >
332   <xs:attribute name="count" use="optional" type="xs:nonNegativeInteger"/>
333   <xs:attribute name="offset" use="optional" type="xs:nonNegativeInteger" />
334 </xs:attributeGroup>
335
336
```

337 **Figure 3. `dp:BasicPagingAttributeGroup` — Schema Fragment**

### 338 3.1.2. Basic Pagination - Response Attributes

339 The attributes defined for the pagination feature on response messages are used to provide the invoking party with the  
340 information that they may need to build the subsequent request. The `dp:BasicPagingResponseAttributeGroup`  
341 attribute group includes the following attributes:

- 342 • `remaining` [**Optional**] - the number of *items* remaining to be read **after** the results returned in this response.  
343 While this attribute is optional in the schema, the attribute **MUST** be specified in the response if the associated  
344 request included either the `count` or `offset` attributes.  
345 The value zero in the `remaining` attribute indicates that there are no further items available for this request (i.e.,  
346 we're done).  
347 The special value -1 in the `remaining` attribute indicates that the service does not have a definitive count of the  
348 number of remaining entries and so the caller should come back for more. This is frequently used for cases where  
349 the caller is reading a variable length stream of data (such as a stock ticker).  
350 The number of items in the result set can change as the result set is read, resulting in a different total number of  
351 items than originally calculated. So, after a read of the first 10 items and with a remaining count of 11, the read of  
352 the next 10 items may get back a remaining count of 2 instead of the expected 1. This can be prevented if static  
353 set processing is supported by the service and invoked by the caller (see [Section 3.2](#)).  
354 • `nextOffset` [**Optional**] - the offset in the result set for the first *item* of the remaining *items*.  
355 While `nextOffset` is defined as optional in the schema, the attribute **MUST** be specified on a response when the  
356 `remaining` attribute is present and has a non-zero value.  
357 The recipient of the response, would place the value of the `nextOffset` attribute into the `offset` attribute in a  
358 subsequent request to read the next page of *items*.  
359 • `maxCount` [**Optional**] - the maximum number of items that the service will return in a single response. This  
360 attribute is normally only specified when a *ResultsTooLarge* error code is being returned (see [Section 3.4](#) below).

```
361 <!-- BasicPagingResponseAttributeGroup - basic response pagination support -->
362
363 <xs:attributeGroup name="BasicPagingResponseAttributeGroup">
364   <xs:attribute name="remaining" use="optional" type="xs:integer"/>
365   <xs:attribute name="nextOffset" use="optional" type="xs:nonNegativeInteger" />
366   <xs:attribute name="maxCount" use="optional" type="xs:nonNegativeInteger" />
367 </xs:attributeGroup>
368
```

369 **Figure 4. dp:BasicPagingResponseAttributeGroup Schema Fragment**

## 370 3.2. Extended Pagination

371 Extended pagination includes all the attributes and features of basic pagination and adds the ability to define and  
372 reference static result sets (so that the results are consistent across multiple read operations). With a static set defined,  
373 the results would be the same as if the requester had read them in a single read operation.

374 Static sets do place a burden on the server to maintain the result set across multiple invocations. This may not even  
375 make sense with certain types of data. Therefore, when adopting pagination for a service definition, the authors  
376 should evaluate whether static sets are necessary and if not, simply use the attributes and features of Basic Pagination  
377 (see [Section 3.1](#) above).

### 378 3.2.1. Extended Pagination - Request attributes

379 Extended pagination builds upon the attributes defined for Basic Pagination. the attribute group  
380 dp:ExtendedPagingAttributeGroup contains all of the attributes (including their rules and interpretations) in  
381 the dp:BasicPagingAttributeGroup as well as the following added attributes:

382 • **setID [Optional]** - the identity of the result set being accessed. This attribute **MUST ONLY** be specified when  
383 the setID was returned on a previous response to a request that established the result set.

384 If this attribute is specified, the normal parameters for the operation **MUST NOT** be specified as the parameters  
385 used when the result set was established control the ongoing responses using this result set. Only the pagination  
386 attributes (and, potentially, an identity attribute for signing purposes) **SHOULD** be present on such requests.

387 • **setReq [Optional]** - a controlling attribute used to indicate the desire for a new set, or ask the server to delete an  
388 established set. This attribute **MUST** have one of the following values:

389 • *Static* - a new static results set is to be created using the parameters specified on this operation. The setID  
390 attribute **MUST NOT** be specified on the request when this value is specified.

391 • *DeleteSet* - the existing result set indicated by the setID attribute are to be deleted. This is usually only  
392 necessary when the caller wants to abort reading the entire result set as the result set is automatically deleted  
393 when the last item is read.

394 The schema for the `dp:ExtendedPagingAttributeGroup` is shown below.

```
395 <!-- ExtendedPagingAttributeGroup - adds set support -->
396
397 <xs:attributeGroup name="ExtendedPagingAttributeGroup" >
398   <xs:attributeGroup ref="BasicPagingAttributeGroup" />
399   <xs:attribute name="setID" use="optional" type="xs:string"/>
400   <xs:attribute name="setReq" use="optional">
401     <xs:simpleType>
402       <xs:restriction base="xs:string">
403         <xs:enumeration value="Static"/>
404         <xs:enumeration value="DeleteSet"/>
405       </xs:restriction>
406     </xs:simpleType>
407   </xs:attribute>
408 </xs:attributeGroup>
409
410
```

411 **Figure 5. `dp:ExtendedPagingAttributeGroup` Schema Fragment**

## 412 3.2.2. Extended Pagination - Response Attributes

413 Like the request attributes defined above, the response attributes for Extended Pagination are built atop the Basic  
414 Pagination response attributes. The attribute group `dp:ExtendedPagingResponseAttributeGroup` contains  
415 all of the attributes (including their rules and interpretations) in the `dp:BasicPagingResonseAttributeGroup`  
416 as well as the following added attributes:

- 417 • `setID` [**Optional**] - the identity of the result set to which these results belong. This attribute **MUST** be specified  
418 when the results are associated with a static result set.
  - 419 • `setExpires` [**Optional**] - the time at which this set will no longer be valid. This attribute **MUST** be specified in  
420 the initial response that creates the `setID` and **MAY** be specified in subsequent responses.
- 421 The WSP is telling the WSC that it will maintain the static result set until this point in time and afterwards requests  
422 using that `setID` will fail.

```
423 <!-- ExtendedPagingResponseAttributeGroup - adds support for sets -->
424
425 <xs:attributeGroup name="ExtendedPagingResponseAttributeGroup">
426   <xs:attributeGroup ref="BasicPagingResponseAttributeGroup" />
427   <xs:attribute name="setID" use="optional" type="xs:string"/>
428   <xs:attribute name="setExpires" use="optional" type="xs:dateTime"/>
429 </xs:attributeGroup>
430
```

431 **Figure 6. `dp:ExtendedPagingResponseAttributeGroup` Schema Fragment**

## 432 3.3. Pagination Examples

### 433 3.3.1. Pagination Schema Example

434 This example shows how the pagination attributes can be included into a schema (and is the schema that we use for  
435 the remaining examples).

```
436 <?xml version="1.0" encoding="UTF-8"?>
437 <xs:schema targetNamespace="urn:liberty:ex:getdata"
438   xmlns:xs="http://www.w3.org/2001/XMLSchema"
439   xmlns:dp="urn:liberty:dp:2007-09"
440   xmlns="urn:liberty:ex:getdata"
441   elementFormDefault="qualified"
442   attributeFormDefault="unqualified"
443 >
444   <xs:import namespace="urn:liberty:dp:2007-09" schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>
445
446   <xs:element name="GetData" type="GetDataType" />
447   <xs:complexType name="GetDataType" mixed="true">
448     <xs:attributeGroup ref="dp:ExtendedPagingAttributeGroup" />
449   </xs:complexType>
450
451   <xs:element name="GetDataResponse" type="GetDataResponseType" />
452   <xs:complexType name="GetDataResponseType" mixed="true">
453     <xs:attributeGroup ref="dp:ExtendedPagingResponseAttributeGroup" />
454   </xs:complexType>
455
456 </xs:schema>
457
458
```

459 **Example 5. GetData Schema incorporating pagination attributes**

### 460 3.3.2. Basic Pagination Example

461 An example sequence of request and response messages using the pagination attributes to read 21 items from a data  
462 service, 10 items at a time.

#### 463 3.3.2.1. 1. Initial request for max of 10 items.

464 This request asks for at most 10 items from the result set (and does not specify an offset, which is the same as  
465 specifying an offset of zero).

```
466 <GetData xmlns="urn:liberty:ex:getdata" count="10">
467   ... get data parameters ...
468 </GetData>
469
```

470 **Example 6. Initial request, max of 10**

#### 471 3.3.2.2. 2. Response to initial request with 11 remaining.

472 This response includes the 10 items requested and specifies that there are 11 items remaining starting at offset 10.

```
473 <GetDataResponse xmlns="urn:liberty:ex:getdata" remaining="11" nextOffset="10">
474   ... data items go here (10 of them) ...
475 </GetDataResponse>
476
```

477 **Example 7. First response, remaining 11**

#### 478 3.3.2.3. 3. Request for 10 more items.

479 This request asks for at most 10 more items from the result set starting at offset 10 (the 11th item).

```
480 <GetData xmlns="urn:liberty:ex:getdata" count="10" offset="10">
481   ... get data parameters ...
482 </GetData>
483
```

484 **Example 8. Second request, max of 10**

#### 485 **3.3.2.4. 4. Response to request with 1 remaining.**

486 This response includes the 10 items requested and specifies that there is 1 more item remaining starting at offset 20.

```
487 <GetDataResponse xmlns="urn:liberty:ex:getdata" remaining="1" nextOffset="20" >
488   ... GetData items go here (10 more)...
489 </GetDataResponse>
490
```

491 **Example 9. Second response, remaining 1**

#### 492 **3.3.2.5. 5. Request for 10 more items.**

493 This request asks for at most 10 more items from the result set starting at offset 20 (the 21st item). Note that given  
494 the prior response specifying that there is only 1 remaining, this request could have asked for just 1 more item. Instead  
495 it chose to ask for 10 in case the count of items had changed.

```
496 <GetData xmlns="urn:liberty:ex:getdata" count="10" offset="20">
497   ... get data parameters ...
498 </GetData>
499
```

500 **Example 10. Third request, max of 10**

#### 501 **3.3.2.6. 6. Response to request with 0 remaining.**

502 This response includes the 1 items remaining and specifies that there are no more remaining items (and as such doesn't  
503 include an next offset).

```
504 <GetDataResponse xmlns="urn:liberty:ex:getdata" remaining="0" >
505   ... GetData items go here (1 this time)...
506 </GetDataResponse>
507
```

508 **Example 11. Third response, remaining 0**

### 509 **3.3.3. Extended Pagination Example**

510 An example sequence of request and response messages using the pagination attributes to read 21 items from a data  
511 service, 10 items at a time.

#### 512 **3.3.3.1. 1. Initial request for max of 10 items and creation of static set.**

513 This request asks for at most 10 items from the result set (and does not specify an offset, which is the same as  
514 specifying an offset of zero). The request also asks the service to create a static set for the remaining operations.

```
515 <GetData xmlns="urn:liberty:ex:getdata" count="10" setReq="Static" >
516   ... get data parameters ...
517 </GetData>
518
```

519 **Example 12. Initial request, max of 10, create static result set**

### 520 3.3.3.2. 2. Response to initial request with 11 remaining.

521 This response includes the 10 items requested, defines the setID for the result set, and specifies that there are 11 items  
522 remaining starting at offset 10.

```
523 <GetDataResponse xmlns="urn:liberty:ex:getdata" setID="38273923" setExpires="2007-01-14T17:33:11Z" remaining="11"
524   ... data items go here (10 of them) ...
525 </GetDataResponse>
526
527
```

528 **Example 13. First response, remaining 11, assigns setID**

### 529 3.3.3.3. 3. Request for 10 more items from static result set.

530 This request asks for at most 10 more items from the specified static result set starting at offset 10 (the 11th item).

531 Note that this request does **not** include request parameters. They were defined when the static result set was created  
532 and are no longer necessary.

```
533 <GetData xmlns="urn:liberty:ex:getdata" setID="38273923" count="10" offset="10" />
534
```

535 **Example 14. Second request, max of 10, uses setID**

### 536 3.3.3.4. 4. Response to request with 1 remaining.

537 This response includes the 10 items requested and specifies that there is 1 more item remaining in the static result set  
538 starting at offset 20.

```
539 <GetDataResponse xmlns="urn:liberty:ex:getdata" setID="38273923" remaining="1" nextOffset="20">
540   ... data items go here (10 more) ...
541 </GetDataResponse>
542
```

543 **Example 15. Second response, remaining 1, uses setID**

### 544 3.3.3.5. 5. Request for 10 more items from static result set.

545 This request asks for at most 1 more item from the static result set starting at offset 20 (the 21st item). Since this is  
546 a static result set, there can't be more than 1 item left.

547 Note that this request does **not** include request parameters. They were defined when the static result set was created  
548 and are no longer necessary.

```
549 <GetData xmlns="urn:liberty:ex:getdata" setID="38273923" count="1" offset="20" />
550
```

551 **Example 16. Third request, max of 1, uses setID**

552 **3.3.3.6. 6. Response to request with 0 remaining.**

553 This response includes the 1 items remaining and specifies that there are no more remaining items (and as such doesn't  
554 include an next offset).

555 Following this response, since the caller has read the entire static result set, the `setID` is no longer valid since the  
556 server will have deleted this set following the read of the last item. Any further attempts to use this `setID` will result  
557 in an error.

```
558 <GetDataResponse xmlns="urn:liberty:ex:getdata" setID="38273923" remaining="0">  
559   ... GetData items go here (1 this time)...  
560 </GetDataResponse>  
561
```

562 **Example 17. Third response, remaining 0**

563 **3.4. Pagination Processing Rules**

564 • If a request includes a `count` attribute, the response **MUST NOT** include more than `count` items. The response  
565 **MAY** include less items, either because there aren't any more items to respond or because the server decided on  
566 its own to return a smaller number – even with more results available.

567 • If a request does **NOT** include a `count` attribute, but the number of items in the result is, at the discretion of the  
568 service, too large to return in a single response, the request **MUST** fail and, if detailed error codes are provided,  
569 the error code **MUST** be *ResultsTooLarge*.

570 In such cases, the service **SHOULD** specify the `maxCount` attribute on the response, indicating the largest result  
571 set it is willing to return. The WSC would, in such cases, resubmit the request with `count` set to this value or a  
572 lesser value.

573 • If a request includes an `offset` attribute, the data in the response must start at the specified offset into the results  
574 defined by the operation's parameters. If this offset is beyond the end of the results set, the request **MUST** fail  
575 and, if detailed error codes are provided, the error code **MUST** be *OffsetBeyondEnd*.

576 • If a request includes a `setReq` attribute with the value "Static," the WSP **MUST** arrange to return a static set of  
577 results using the current request parameters and **MUST** identify this set with a unique identifier specified in the  
578 `setID` of the response. This does **not** require a particular implementation of the static results (e.g., some WSPs  
579 could cache the results, others could use backend database cursor capabilities).

580 • Following the creation of a static results set, the WSC **SHOULD NOT** specify search criteria on subsequent  
581 requests related to the same set. If this criteria is specified, the WSP **MAY** return or a failure or otherwise **MUST**  
582 ignore it and use the search criteria specified when the static result set was created. By search criteria, we mean  
583 any operational parameters to the request that are used to control the results set (other than pagination attributes,  
584 of course).

585 • The WSP **MUST** ensure that static result sets created by requests from one WSC are not made visible to other  
586 WSCs.

587 • If a request includes a `setID` that is not valid (because it was not generated by that WSP, not assigned to the  
588 requesting WSC, or refers to a set which has been deleted), the request **MUST** fail and, if detailed error codes are  
589 provided, the error code **MUST** be *StaticSetInvalid*.

## 590 **4. Delayed Notification**

591 Some service interfaces provide for a delayed completion of their processing. This can occur for several reasons  
592 including delayed operation (a request with a future timestamp) and/or indirect operation (a request that is forwarded  
593 to another party for processing).

594 Such delayed operation raises an issue for the service in that they need a means to provide the invoker with the  
595 completion status and results of the operation but do not want a request to hang about on their service interface for  
596 long periods of time (tying up valuable resources).

597 At the same time, there are many cases where the service can determine if there is a problem with the request right  
598 away and return such status in an immediate response to the service invocation.

599 This design pattern solves the issues by providing:

- 600 1. a means for the invoker to use to indicate that they are able to receive delayed results and where to send such  
601 delayed (Section 4.2.1).
- 602 2. a means for immediate results to be returned, if available.
- 603 3. a means to indicate that immediate results are not available and will be returned later.
- 604 4. a means to deliver the delayed results to the invoker (including the definition of an interface that must be exposed  
605 by the invoker to receive such delayed results).

### 606 **4.1. Delayed Notification Sequence**

607 The following sections describe an example sequence of events that is representative of a typical implementation of  
608 this design pattern. Of course, this is not a required sequence and some implementations and/or particular invocations  
609 within a given implementation will result in different sequences of steps. However, this example does explain the  
610 potential steps/processing that may take place.

#### 611 **4.1.1. Step 1: Invoker submits request with <dp:NotifyTo> element**

612 If the invoker supports and wants delayed notification for the completion status, they would include a <dp:NotifyTo>  
613 element on the request which contains the [WSAv1.0] EndpointReference (EPR) for their notification endpoint.

614 The presence of this element indicates that they want the results of any delayed operation and identifies the location to  
615 which such delayed operation results should be sent.

616 If this element was not present in the request, the processing rules for the service interface MUST define the behavior  
617 of the interface. This will typically fall into one of the following three options:

- 618 • The <dp:NotifyTo> element is required and the request fails if not specified.
- 619 • The service makes best efforts to verify that the future/indirect processing of the request will succeed and return the  
620 results of those efforts and accept that a later failure may occur without the ability to notify, directly, the invoker.
- 621 • The service waits for the completion of the process prior to returning the actual completion status and/or results to  
622 the invoker. This is more likely to be used in an indirect operation.

623 The service specification MUST document which option is chosen for this case. In many cases the "best efforts"  
624 solution is the simplest and probably best for such cases as the invoker is clearly indicating that they do not want the  
625 results sent back separately.

#### 626 **4.1.2. Step 2: Service instance performs initial processing**

627 The service should process and validate the results to the extent possible at this time. The service has the choice of  
628 returning one of the following status codes for each request item:

629 • *OK* - the validation is complete for this item and the update will proceed as indicated (if a future operation) or has  
630 completed successfully (if an indirect operation and/or an operation with a past timestamp).

631 For a delayed operation, the service is indicating that the service will complete without error (as in they know  
632 that it will be successful). Since this is hard to ensure with reasonable soundness for future operations, it is not  
633 recommended that this status be returned until the operation has actually completed.

634 If this status is returned, no further messages will be sent related to this request item.

635 • *WillNotify* - the request has been validated to the extent possible by the service instance and will be processed as  
636 requested. The completion status of the request will be sent to the invoker when the processing is complete.

637 • *anything else* - any other status value indicates that the validation and/or processing for this item has failed (the  
638 operation was not successful).

639 If this is returned, no further messages will be sent related to this request item.

640 The inclusion of multiple request items in a single request is possible in many service interfaces. In some cases, the  
641 multiple operations are treated as an atomic operation and therefore the status codes above apply to the single atomic  
642 operation.

643 In other cases, the service interface allows the individual request items to be processed independently and for partial  
644 results to be returned. In such cases, the secondary status codes for each request item in partial operations would  
645 meet the rules outlined above and the notification messages would be individual for each request item. For example,  
646 a request for 5 operations could result in 2 failing validation immediately and 3 separate notifications of completion,  
647 one for each other request item in the request.

#### 648 **4.1.3. Step 3: The operation completes**

649 Upon completion of an operation, if the status code *WillNotify* was returned for that operation, the service instance  
650 must generate a `<dp:Notification>` message and send it to the invoker at the destination indicated in the  
651 `<dp:NotifyTo>` element that was on the request.

652 The `<dp:Notification>` message MUST contain the application response message which contains the completion  
653 status of the request as well as any possible results.

## 654 **4.2. Delayed Notification data structures**

### 655 **4.2.1. `<dp:NotifyTo>` Element**

656 The `<dp:NotifyTo>` element is an ID-WSF `EndpointReference` (see [[LibertyDisco](#)]) which describes where com-  
657 pletion notification messages are to be sent.

658 In the case where a `<dp:NotifyTo>` is included in a service request that is part of a polling response message,  
659 the anonymous address (<http://www.w3.org/2005/08/addressing/anonymous>) may be used in the `<dp:NotifyTo>` to  
660 indicate that the notification message is to be sent to the same endpoint that the poll request was submitted.

661 Service specifications which adopt this design pattern MUST include this element in the schema definitions for the  
662 interfaces where the capabilities of delayed notification are desired.

663 The schema for the <dp:NotifyTo> element is shown below.

```
664 <!-- NotifyTo - element for carrying the notification destination -->
665
666 <xs:element name="NotifyTo" type="wsa:EndpointReferenceType" />
667
```

668 **Figure 7. <dp:NotifyTo> — Schema Fragment**

669 An example usage of the <dp:NotifyTo> element in a service schema for the Liberty ID-WSF Provisioning Service  
670 ([LibertyPROV]) <prov:PMUpdate> interface (an indirect operation) is shown below:

```
671 <!-- PMUpdate - update the PM for a existing PM at the ProvS -->
672
673 <xs:element name="PMUpdate" type="PMUpdateType"/>
674
675 <xs:complexType name="PMUpdateType">
676   <xs:complexContent>
677     <xs:extension base="RequestAbstractType">
678       <xs:sequence>
679         <xs:element ref="PMUpdateItem" maxOccurs="unbounded" />
680         <xs:element ref="dp:NotifyTo" minOccurs="0" />
681       </xs:sequence>
682     </xs:extension>
683   </xs:complexContent>
684 </xs:complexType>
685
686 <xs:element name="PMUpdateItem" type="PMUpdateItemType" />
687
688 <xs:complexType name="PMUpdateItemType">
689   <xs:sequence>
690     <xs:element ref="PMDescriptor"/>
691   </xs:sequence>
692   <xs:attribute name="type" type="xs:anyURI" use="required"/>
693   <xs:attribute name="itemID" type="xs:string" use="required"/>
694   <xs:attribute name="at" type="xs:dateTime" use="optional"/>
695 </xs:complexType>
696
```

697 **Example 18. Example schema inclusion of <dp:NotifyTo> element**

698 An example message which includes the <dp:NotifyTo> element.

```
699 <prov:PMUpdate>
700   <prov:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
701     <prov:PMDescriptor xs:id="2323923900239" >
702       <prov:PMID issuer="http://provs-r-us.com">uuid:778349-283920-88379-5448739</prov:PMID>
703       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
704       <ds:Signature>
705         ... signature data goes here ...
706       </ds:Signature>
707     </prov:PMDescriptor>
708   </prov:PMUpdateItem>
709   <dp:NotifyTo>
710     <wsa:Address>https://provider.com/notifications</wsa:Address>
711     <wsa:Metadata>
712       <ds:ProviderID>http://provider.com/</ds:ProviderID>
713       <ds:ServiceType>urn:liberty:dp:2007-09:notification</ds:ServiceType>
714       <ds:Framework version="2.0" />
715       <ds:SecurityContext>
716         <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:null</ds:SecurityMechID>
717       </ds:SecurityContext>
718     </wsa:Metadata>
719   </dp:NotifyTo>
720 </prov:PMUpdate>
721
```

722 **Example 19. Example message inclusion of `<dp:NotifyTo>` element**

## 723 **4.3. Delayed Notification Operations**

### 724 **4.3.1. Operation: *Notification***

725 The *Notification* operation is reverse channel service interface exposed by a web services consumer (WSC) acting as a  
726 web service provider (WSP) in order to receive a delayed notification message from another service provider that the  
727 WSC had invoked.

#### 728 **4.3.1.1. `wsa:Action` values for Notification Messages**

729 `<Notification>` request messages **MUST** include a `<wsa:Action>` SOAP header with the value of  
730 "urn:liberty:dp:2007-09:Notification."

731 `<NotificationResponse>` messages **MUST** include a `<wsa:Action>` SOAP header with the value of  
732 "urn:liberty:dp:2007-09:NotificationResponse."

#### 733 **4.3.1.2. Notification Message**

734 The `<Notification>` request is called to send a delayed completion response to a provider.

735 The `<dp:Notification>` request contains one or more service level response messages associated with a prior  
736 service request at the invoker's WSP. Note that for the `<dp:Notification>` message, the invoker is the recipient of  
737 the referenced message and the recipient of this `<dp:Notification>` message is the invoker of the former message  
738 that caused this notification to be sent.

739 In other words, the two parties have switched roles.

740 The `<dp:Notification>` element contains the following elements and attributes:

741 • `<xs:any>` **[Required]** - a catch-all element to allow the insertion of a service response message (the complete  
742 contents of what would normally be in the body of a response message to a service request).

743 For example, if this message was a delayed notification for the completion of a `<PMDUpdate>` request, this element  
744 would be a `<PMDUpdateResponse>`.

745 The contents of this element are controlled by the service specification which contains the request that is being  
746 responded to. Typically, Liberty specifications require that the body of a response have exactly one element and  
747 so there will typically be exactly one element in this location.

748 • `ref` **[Required]** - the message ID from the request message which resulted in this notification being sent.

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

751 The schema for the `<dp:Notification>` is shown below.

```
752 <!-- Notification - interface for receiving the delayed completion status -->
753
754 <xs:element name="Notification" type="NotificationType"/>
755
756 <xs:complexType name="NotificationType">
757   <xs:complexContent>
758     <xs:extension base="RequestAbstractType">
759       <xs:sequence>
760         <xs:any namespace="##other"
761           processContents="lax"
762           maxOccurs="unbounded" />
763       </xs:sequence>
764       <xs:attribute name="ref" type="xs:string" use="required" />
765     </xs:extension>
766   </xs:complexContent>
767 </xs:complexType>
768
```

769 **Figure 8. `<dp:Notification>` — Schema Fragment**

770 An example message body containing a `<dp:Notification>` message follows. This contains a completion  
771 notification for a `<prov:PMDUpdate>` operation.

```
772 <dp:Notification ref="...messageID-of-request...">
773   <prov:PMUpdateResponse>
774     <lu:Status code="OK"/>
775   </prov:PMUpdateResponse>
776 </dp:Notification>
777
```

778 **Example 20. Example `<dp:Notification>` Message**

### 779 4.3.1.3. NotificationResponse Message

780 This response to the `<dp:Notification>` request contains the following elements/attributes:

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

```
785 <!-- NotificationResponse - the response to a Notification message -->
786
787 <xs:element name="NotificationResponse" type="NotificationResponseType" />
788
789 <xs:complexType name="NotificationResponseType">
790   <xs:complexContent>
791     <xs:extension base="ResponseAbstractType" />
792   </xs:complexContent>
793 </xs:complexType>
794
```

795 **Figure 9. <dp:NotificationResponse> — Schema Fragment**

796 An example message body containing a <NotificationResponse> message follows.

```
797 <dp:NotificationResponse>
798   <lu:Status code="OK" />
799 </dp:NotificationResponse>
800
```

801 **Example 21. Example <dp:NotificationResponse> Message**

#### 802 4.3.1.4. Notification Processing Rules

- 803 • If the recipient is unable to locate a pending request with the message id specified in the `ref` attribute, the call  
804 MUST be treated as a failure. In such cases, if detailed status codes are being included, the detailed status code for  
805 this error MUST be *NotFound*.
- 806 • Each <dp:Notification> MUST ONLY include response data for a single request (the sender MAY NOT  
807 combine results from different requests).
- 808 • When building a notification message, if there are still outstanding operations for which the completion data or  
809 status is not available, the sender MUST indicate the current status of those items using the "WillNotify" status  
810 code.
- 811 • For multiple item requests that allow partial results, the sender MAY include the status of some or all of the pending  
812 items on a notification. So a single request with multiple request items could result in a single delayed notification,  
813 several delayed notifications, or even one delayed notification for each and every request item.
- 814 • When multiple item requests are supported, the same method used to match results in the non-delayed response  
815 MUST be used to match results in the delayed notification. Typically, the request items from the request would  
816 have an identifier that is placed to a `ref` attribute in the service level response item (note that this is different from  
817 the `ref` attribute in the <dp:NotificationResponse> element).
- 818 On the other hand, if the normal service request uses order of elements in the response to match them to the request  
819 items, the sender MUST wait to accumulate the results for all items prior to returning any. This method is NOT  
820 RECOMMENDED.
- 821 • If request processing succeeded, the top-level status code MUST be "OK." Otherwise, the top-level status code  
822 MUST be "Failed."
- 823 • If the top-level status code is "Failed," the response MAY also contain *NotFound* as a second-level status code.  
824 The service instance may not wish to reveal the reason for failure, in which case no second-level status code will  
825 appear.

---

## 826 **4.4. Delayed Notification Examples (Non-Normative)**

827 This section walks through a fictitious sequence of events in a delayed notification environment. This sequence is  
828 made particularly complex in order to highlight the possible messages one could observe while most real-world usages  
829 are likely to be simpler.

### 830 **4.4.1. Step 1: Single request to update several PMDs**

831 This request involves an update of several PMDs that have previously been provisioned to a multitude of PMMs in  
832 different locations. Since this is an indirect operation through the Provisioning Service that may not complete right  
833 away, the caller includes a `<dp:NotifyTo>` element in the request.

```
834 <prov:PMUpdate>
835   <prov:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
836     <prov:PMDescriptor xs:id="2323923900239" >
837       <prov:PMID issuer="http://provs-r-us.com">uuid:cf5fab69-3092-4ef3-a7c0-f97e70ad769b<
838 /prov:PMID>
839       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
840     </prov:PMDescriptor>
841   </prov:PMUpdateItem>
842   <prov:PMUpdateItem itemID="2" type="urn:liberty:prov:2007-09:ut:engine">
843     <prov:PMDescriptor xs:id="2323923900239" >
844       <prov:PMID issuer="http://provs-r-us.com">uuid:20643542-3f8c-4281-a4a6-
845 54b07d07ddab</prov:PMID>
846       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
847     </prov:PMDescriptor>
848   </prov:PMUpdateItem>
849   <prov:PMUpdateItem itemID="3" type="urn:liberty:prov:2007-09:ut:engine">
850     <prov:PMDescriptor xs:id="2323923900239" >
851       <prov:PMID issuer="http://provs-r-us.com">uuid:ee059d84-8819-48c9-a46d-f7fbbc52866c</pro
852 v:PMID>
853       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
854     </prov:PMDescriptor>
855   </prov:PMUpdateItem>
856   <prov:PMUpdateItem itemID="4" type="urn:liberty:prov:2007-09:ut:engine">
857     <prov:PMDescriptor xs:id="2323923900239" >
858       <prov:PMID issuer="http://provs-r-us.com">uuid:bf3cbd82-23c2-486e-b34b-301f
859 07827b61</prov:PMID>
860       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
861     </prov:PMDescriptor>
862   </prov:PMUpdateItem>
863   <prov:PMUpdateItem itemID="5" type="urn:liberty:prov:2007-09:ut:engine">
864     <prov:PMDescriptor xs:id="2323923900239" >
865       <prov:PMID issuer="http://provs-r-us.com">uuid:7091b77c-b7e7-42d0-8fba-d6ddac45991e</prov:P
866 MID>
867       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
868     </prov:PMDescriptor>
869   </prov:PMUpdateItem>
870   <prov:PMUpdateItem itemID="6" type="urn:liberty:prov:2007-09:ut:engine">
871     <prov:PMDescriptor xs:id="2323923900239" >
872       <prov:PMID issuer="http://provs-r-us.com">uuid:e711fd21-10ba-41a9-bf46-5dfed4c0
873 441a</prov:PMID>
874       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
875     </prov:PMDescriptor>
876   </prov:PMUpdateItem>
877 <dp:NotifyTo>
878   <wsa:Address>https://provider.com/notifications</wsa:Address>
879   <wsa:Metadata>
880     <ds:ProviderID>http://provider.com/</ds:ProviderID>
881     <ds:ServiceType>urn:liberty:dp:2007-09:notification</ds:ServiceType>
882     <ds:Framework version="2.0" />
883     <ds:SecurityContext>
884       <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:null</ds:SecurityMechID>
885     </ds:SecurityContext>
886   </wsa:Metadata>
887 </dp:NotifyTo>
888 </prov:PMUpdate>
889
```

#### 890 **Example 22. Step 1: request to update several PMDs**

891 This request is attempting to update 6 different PMs to the same new version of a pmengine.

#### 892 **4.4.2. Step 2: Initial response from the Provisioning Service**

893 Of the 6 requested items, the Provisioning service is able to process, immediately, one of them and one of them fails.

894 The service returns the partial results shown below.

```
895 <prov:PMUpdateResponse>
896   <lu:Status code="Partial">
897     <lu:Status ref="1" code="WillNotify" />
898     <lu:Status ref="2" code="OK" />
899     <lu:Status ref="3" code="WillNotify" />
900     <lu:Status ref="4" code="WillNotify" />
901     <lu:Status ref="5" code="NotFound" />
902     <lu:Status ref="6" code="WillNotify" />
903   </lu:Status>
904 </prov:PMUpdateResponse>
905
```

#### 906 **Example 23. Step 2: Initial response**

907 A possible alternative at this stage is that none of the results are currently available. In this case, the response would  
908 have returned a primary status code of "WillNotify." An example of such a return is below.

```
909 <prov:PMUpdateResponse>
910   <lu:Status code="WillNotify" />
911 </prov:PMUpdateResponse>
912
```

#### 913 **Example 24. Initial response with no completed operations**

### 914 **4.4.3. Step 3: First Notification**

915 After some time, one of the remaining 4 items completes processing and a notification is sent. Note that the special  
916 status "Notify," which indicates that this is a limited status report that only includes the remaining outstanding items  
917 (some of which are still in the "WillNotify" state), is used.

```
918 <dp:Notification ref="uuid:2376bf6d-9cf6-420e-a438-436a30f7d3f1" >
919   <prov:PMUpdateResponse>
920     <lu:Status code="Notify">
921       <lu:Status ref="1" code="WillNotify" />
922       <lu:Status ref="3" code="OK" />
923       <lu:Status ref="4" code="WillNotify" />
924       <lu:Status ref="6" code="WillNotify" />
925     </lu:Status>
926   </prov:PMUpdateResponse>
927 </dp:Notification>
928
```

#### 929 **Example 25. Step 3: First Notification**

### 930 **4.4.4. Step 4: Second Notification**

931 After some additional time, two of the remaining 3 items completes processing and a notification is sent. Note that the  
932 special status "Notify," which indicates that this is a limited status report that only includes the remaining outstanding  
933 items (the last of which is still in the "WillNotify" state), is used.

```
934 <dp:Notification ref="uuid:2376bf6d-9cf6-420e-a438-436a30f7d3f1" >
935   <prov:PMUpdateResponse>
936     <lu:Status code="Notify">
937       <lu:Status ref="1" code="OK" />
938       <lu:Status ref="4" code="Failed" />
939       <lu:Status ref="6" code="WillNotify" />
940     </lu:Status>
941   </prov:PMUpdateResponse>
942 </dp:Notification>
943
```

944

**Example 26. Step 4: Second Notification**

945 **4.4.5. Step 5: Final Notification**

946 After some additional time, the last remaining item completes processing and a notification is sent. Note that the  
947 special status "*Notify*," which indicates that this is a limited status report that only includes the remaining outstanding  
948 items (and no other items are still outstanding since none are still in the "*WillNotify*" state), is used.

```
949 <dp:Notification ref="uuid:2376bf6d-9cf6-420e-a438-436a30f7d3f1" >  
950   <prov:PMUpdateResponse>  
951     <lu:Status code="Notify">  
952       <lu:Status ref="6" code="OK" />  
953     </lu:Status>  
954   </prov:PMUpdateResponse>  
955 </dp:Notification>  
956
```

957

**Example 27. Step 5: Final Notification**

## 958 5. ID-WSF Design Patterns Schema

```

959 <?xml version="1.0" encoding="UTF-8"?>
960 <xs:schema targetNamespace="urn:liberty:dp:2007-09"
961   xmlns:lu="urn:liberty:util:2006-08"
962   xmlns:xs="http://www.w3.org/2001/XMLSchema"
963   xmlns:wsa="http://www.w3.org/2005/08/addressing"
964   xmlns="urn:liberty:dp:2007-09"
965   elementFormDefault="qualified"
966   attributeFormDefault="unqualified"
967 >
968
969 <xs:import namespace="urn:liberty:util:2006-08"
970   schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
971
972 <xs:import namespace="http://www.w3.org/2005/08/addressing"
973   schemaLocation="http://www.w3.org/2005/08/addressing/ws-addr.xsd" />
974
975 <!--*****-->
976 <!--      -->
977 <!--      Polling design pattern schema defs      -->
978 <!--      -->
979 <!--*****-->
980
981
982 <!-- PollType - datatype for polling the recipient for any new work -->
983
984 <xs:complexType name="PollType">
985   <xs:sequence>
986     <xs:element ref="wsa:Action" minOccurs="0" maxOccurs="unbounded" />
987     <xs:element ref="Response" minOccurs="0" maxOccurs="unbounded" />
988   </xs:sequence>
989   <xs:attribute name="wait" type="xs:integer" use="required" />
990   <xs:anyAttribute namespace="##other" processContents="lax" />
991 </xs:complexType>
992
993 <xs:element name="Response" type="ResponseType" />
994 <xs:complexType name="ResponseType">
995   <xs:sequence>
996     <xs:any namespace="##other"
997       processContents="lax"
998       minOccurs="0"
999       maxOccurs="unbounded" />
1000   </xs:sequence>
1001   <xs:attribute name="ref" type="xs:string" use="required" />
1002 </xs:complexType>
1003
1004
1005 <!-- PollResponseType - the datatype of response to a polling message -->
1006
1007 <xs:complexType name="PollResponseType">
1008   <xs:complexContent>
1009     <xs:extension base="ResponseAbstractType">
1010       <xs:sequence>
1011         <xs:element ref="Request" minOccurs="0" maxOccurs="unbounded" />
1012       </xs:sequence>
1013       <xs:attribute name="nextPoll" type="xs:integer" use="optional" />
1014     </xs:extension>
1015   </xs:complexContent>
1016 </xs:complexType>
1017
1018 <xs:element name="Request" type="RequestType" />
1019 <xs:complexType name="RequestType">
1020   <xs:sequence>
1021     <xs:any namespace="##other"
1022       processContents="lax"
1023       minOccurs="0"

```

```

1024         maxOccurs="unbounded" />
1025     </xs:sequence>
1026     <xs:attribute name="itemID" type="xs:string" use="required" />
1027 </xs:complexType>
1028
1029
1030 <!--*****-->
1031 <!-- -->
1032 <!--   Pagination of results design pattern schema defs   -->
1033 <!-- -->
1034 <!--*****-->
1035
1036
1037 <!--BasicPagingAttributeGroup - basic request pagination support -->
1038
1039 <xs:attributeGroup name="BasicPagingAttributeGroup" >
1040     <xs:attribute name="count" use="optional" type="xs:nonNegativeInteger"/>
1041     <xs:attribute name="offset" use="optional" type="xs:nonNegativeInteger" />
1042 </xs:attributeGroup>
1043
1044
1045
1046 <!-- BasicPagingResponseAttributeGroup - basic response pagination support -->
1047
1048 <xs:attributeGroup name="BasicPagingResponseAttributeGroup">
1049     <xs:attribute name="remaining" use="optional" type="xs:integer"/>
1050     <xs:attribute name="nextOffset" use="optional" type="xs:nonNegativeInteger" />
1051     <xs:attribute name="maxCount" use="optional" type="xs:nonNegativeInteger" />
1052 </xs:attributeGroup>
1053
1054
1055 <!-- ExtendedPagingAttributeGroup - adds set support -->
1056
1057 <xs:attributeGroup name="ExtendedPagingAttributeGroup" >
1058     <xs:attributeGroup ref="BasicPagingAttributeGroup" />
1059     <xs:attribute name="setID" use="optional" type="xs:string"/>
1060     <xs:attribute name="setReq" use="optional">
1061         <xs:simpleType>
1062             <xs:restriction base="xs:string">
1063                 <xs:enumeration value="Static"/>
1064                 <xs:enumeration value="DeleteSet"/>
1065             </xs:restriction>
1066         </xs:simpleType>
1067     </xs:attribute>
1068 </xs:attributeGroup>
1069
1070
1071
1072 <!-- ExtendedPagingResponseAttributeGroup - adds support for sets -->
1073
1074 <xs:attributeGroup name="ExtendedPagingResponseAttributeGroup">
1075     <xs:attributeGroup ref="BasicPagingResponseAttributeGroup" />
1076     <xs:attribute name="setID" use="optional" type="xs:string"/>
1077     <xs:attribute name="setExpires" use="optional" type="xs:dateTime"/>
1078 </xs:attributeGroup>
1079
1080
1081 <!--*****-->
1082 <!-- -->
1083 <!--   Delayed Notification design pattern schema defs   -->
1084 <!-- -->
1085 <!--*****-->
1086
1087
1088 <!-- NotifyTo - element for carrying the notification destination -->
1089
1090 <xs:element name="NotifyTo" type="wsa:EndpointReferenceType" />

```

```
1091
1092
1093 <!-- Notification - interface for receiving the delayed completion status -->
1094
1095 <xs:element name="Notification" type="NotificationType"/>
1096
1097 <xs:complexType name="NotificationType">
1098   <xs:complexContent>
1099     <xs:extension base="RequestAbstractType">
1100       <xs:sequence>
1101         <xs:any namespace="##other"
1102           processContents="lax"
1103           maxOccurs="unbounded" />
1104       </xs:sequence>
1105       <xs:attribute name="ref" type="xs:string" use="required" />
1106     </xs:extension>
1107   </xs:complexContent>
1108 </xs:complexType>
1109
1110 <!-- NotificationResponse - the response to a Notification message -->
1111
1112 <xs:element name="NotificationResponse" type="NotificationResponseType" />
1113
1114 <xs:complexType name="NotificationResponseType">
1115   <xs:complexContent>
1116     <xs:extension base="ResponseAbstractType" />
1117   </xs:complexContent>
1118 </xs:complexType>
1119
1120 <!-- RequestAbstractType - common request message structure -->
1121
1122 <xs:complexType name="RequestAbstractType" abstract="true">
1123   <xs:anyAttribute namespace="##other" processContents="lax"/>
1124 </xs:complexType>
1125
1126 <!-- ResponseAbstractType - common message response structure -->
1127
1128 <xs:complexType name="ResponseAbstractType" abstract="true">
1129   <xs:sequence>
1130     <xs:element ref="lu:Status"/>
1131   </xs:sequence>
1132   <xs:anyAttribute namespace="##other" processContents="lax"/>
1133 </xs:complexType>
1134
1135 </xs:schema>
1136
1137
1138
1139
1140
```

## 1141 6. Notification Endpoint WSDL (non-normative)

```
1142 <?xml version="1.0"?>
1143 <definitions name="notify-svc"
1144   targetNamespace="urn:liberty:notify:2007-09"
1145   xmlns:tns="urn:liberty:notify:2007-09"
1146   xmlns="http://schemas.xmlsoap.org/wsdl/"
1147   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1148   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
1149   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
1150   xmlns:dp="urn:liberty:dp:2007-09"
1151   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1152   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
1153     http://schemas.xmlsoap.org/wsdl/
1154     http://www.w3.org/2006/02/addressing/wsdl
1155     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
1156
1157   <types>
1158     <xsd:schema>
1159       <xsd:import namespace="urn:liberty:dp:2007-09"
1160         schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>
1161     </xsd:schema>
1162   </types>
1163
1164   <message name="Notification">
1165     <part name="body" element="dp:Notification"/>
1166   </message>
1167   <message name="NotificationResponse">
1168     <part name="body" element="dp:NotificationResponse"/>
1169   </message>
1170
1171   <portType name="NotifyPort">
1172
1173     <operation name="Notification">
1174       <input message="tns:Notification"
1175         wsaw:Action="urn:liberty:dp:2007-09:Notification" />
1176       <output message="tns:NotificationResponse"
1177         wsaw:Action="urn:liberty:dp:2007-09:NotificationResponse" />
1178     </operation>
1179
1180   </portType>
1181
1182   <!--
1183   An example of a binding and service that can be used with this
1184   abstract service description is provided below.
1185   -->
1186
1187   <binding name="NotifyBinding" type="tns:NotifyPort">
1188
1189     <soap:binding style="document"
1190       transport="http://schemas.xmlsoap.org/soap/http"/>
1191
1192     <operation name="Notification">
1193       <input> <soap:body use="literal"/> </input>
1194       <output> <soap:body use="literal"/> </output>
1195     </operation>
1196
1197   </binding>
1198
1199   <service name="NotifyService">
1200
1201     <port name="NotifyPort" binding="tns:NotifyBinding">
1202
1203       <!-- Modify with the REAL SOAP endpoint -->
1204
1205       <soap:address location="http://example.com/notify"/>
1206
```

```
1207     </port>
1208
1209     </service>
1210
1211 </definitions>
1212
```

## 1213 References

### 1214 Normative

- 1215 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-  
1216 errata-v1.0, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>
- 1217 [LibertyPROV] Cahill, Conor P., eds. "Liberty ID-WSF Provisioning Service Specification," Version 1.0, Liberty  
1218 Alliance Project (14 December 2007). <http://www.projectliberty.org/specs>
- 1219 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"  
1220 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 1221 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-  
1222 errata-v1.0, Liberty Alliance Project (21 April, 2007). <http://www.projectliberty.org/specs>
- 1223 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version  
1224 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1225 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14  
1226 December, 2004). <http://www.projectliberty.org/specs>
- 1227 [LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,  
1228 Single Sign-On, and Identity Mapping Services Specification," v2.0-errata-1.0-01, Liberty Alliance Project  
1229 (28 November, 2006). <http://www.projectliberty.org/specs>
- 1230 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty  
1231 ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007).  
1232 <http://www.projectliberty.org/specs>
- 1233 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet  
1234 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 1235 [RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251,  
1236 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2251.txt>
- 1237 [RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A.  
1238 Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force  
1239 <http://www.ietf.org/rfc/rfc2252.txt>
- 1240 [RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search  
1241 Results," RFC 2891, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2891.txt>
- 1242 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Pro-  
1243 tocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS  
1244 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/>1245 open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)
- 1246 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions  
1247 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-  
1248 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/>1249 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 1250 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,  
1251 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"

- 1252 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards  
1253 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- 1254 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October  
1255 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium  
1256 <http://www.w3.org/TR/xmlschema-1/>
- 1257 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman,  
1258 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C  
1259 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 1260 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).  
1261 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium  
1262 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 1263 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and  
1264 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 1265 [XPath] Clark, J., DeRose, S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0,"  
1266 Recommendation, W3C <http://www.w3.org/TR/xpath> [August 2003].
- 1267 [WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds.  
1268 World Wide Web Consortium W3C Recommendation (9 May 2006). <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/>
- 1269

## 1270 Informative

- 1271 [LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty  
1272 Alliance Project (13 January, 2005). <http://www.projectliberty.org/specs>
- 1273 [LibertyIDPPGuide] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Implementation  
1274 Guidelines," Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 1275 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework  
1276 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>