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

Notice

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

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

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

Contents

1. Introduction

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

1.1. Notation and Conventions

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

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

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

1.1.1. XML Namespaces

The following XML namespaces are referred to in this document:

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

  urn:liberty:dp:2007-09

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

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

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

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

2. Polling

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

2.1. `PollType` Message Type

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

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

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

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

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

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

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

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

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

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

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

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

```
<!-- PollType - datatype for polling the recipient for any new work -->

<xs:complexType name="PollType">
  <xs:sequence>
    <xs:element ref="wsa:Action" minOccurs="0"  maxOccurs="unbounded" />
    <xs:element ref="Response"   minOccurs="0"  maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="wait" type="xs:integer" use="required" />
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

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

Figure 1. <dp:PollType> — Schema Fragment

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

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

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

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

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

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

2.2. `PollResponseType` Message Type

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

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

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

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

```
<!-- PollResponseType - the datatype of response to a polling message -->

<xs:complexType name="PollResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="Request" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="nextPoll" type="xs:integer" use="optional" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

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

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

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

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

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

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

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

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

2.3. Poll Processing Rules

- If the request includes one or more `<wsa:Action>` elements, the recipient MUST NOT include request elements in the response that are not associated with the specified action value(s).

- If the request includes the `wait` attribute, the recipient SHOULD respond immediately when it has requests for the invoker (including any that arrive after the recipient has started waiting for new requests). The recipient MAY use the specified amount of time on the `wait` to determine how long it is willing to wait before responding with no request.

- A single poll response SHOULD NOT include multiple requests which make use of Delayed Notification (Section 4) as this can result in indeterminate results when the results are subsequently returned in different notification using the same `ref` attribute value.

  Requests making use of Delayed Notification SHOULD be sent in separate poll response messages.

- If the response is to be sent without an embedded request, the response should still be treated as a successful response. The lack of a `<dp:Request>` is sufficient to determine that there is no request and the poll operation itself was successful.

- If the response includes a `nextPoll` attribute, the invoker SHOULD NOT poll the recipient again until that amount of time has passed.

  The `nextPoll` SHOULD NOT be specified on responses that include an incoming request as the invoker should issue a new poll containing the response to the request as soon as it has completed processing the request. If the attribute is specified in such cases, the invoker MAY ignore it.

  If a poll is received too soon following a response with a `nextPoll` attribute, the recipient MAY reject the request and in doing so, MAY indicate a secondary status code of "*PollTooSoon*."

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

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

3. Pagination

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

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

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

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

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

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

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

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

3.1. Basic Pagination

3.1.1. Basic Pagination - Request attributes

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

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

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

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

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

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

```
<!--BasicPagingAttributeGroup - basic request pagination support -->

<xs:attributeGroup name="BasicPagingAttributeGroup" >
  <xs:attribute name="count" use="optional"  type="xs:nonNegativeInteger"/>
  <xs:attribute name="offset" use="optional" type="xs:nonNegativeInteger" />
</xs:attributeGroup>
```

Figure 3. `dp:BasicPagingAttributeGroup` — Schema Fragment

3.1.2. Basic Pagination - Response Attributes

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

- `remaining` **[Optional]** - the number of *item*s remaining to be read **after** the results returned in this response.

  While this attribute is optional in the schema, the attribute MUST be specified in the response if the associated request included either the `count` or `offset` attributes.

  The value zero in the `remaining` attribute indicates that there are no further items available for this request (1.e., we're done).

  The special value -1 in the `remaining` attribute indicates that the service does not have a definitive count of the number of remaining entries and so the caller should come back for more. This is frequently used for cases where the caller is reading a variable length stream of data (such as a stock ticker).

  The number of items in the result set can change as the result set is read, resulting in a different total number of items than originally calculated. So, after a read of the first 10 items and with a remaining count of 11, the read of the next 10 items may get back a remaining count of 2 instead of the expected 1. This can be prevented if static set processing is supported by the service and invoked by the caller (see Section 3.2).

- `nextOffset` **[Optional]** - the offset in the result set for the first *item* of the remaining *item*s.

  While `nextOffset` is defined as optional in the schema, the attribute MUST be specified on a response when the `remaining` attribute is present and has a non-zero value.

  The recipient of the response, would place the value of the `nextOffset` attribute into the `offset` attribute in a subsequent request to read the next page of *item*s.

- `maxCount` **[Optional]** - the maximum number of items that the service will return in a single response. This attribute is normally only specified when a *ResultsTooLarge* error code is being returned (see Section 3.4 below).

```
<!-- BasicPagingResponseAttributeGroup - basic response pagination support -->

<xs:attributeGroup name="BasicPagingResponseAttributeGroup">
  <xs:attribute name="remaining" use="optional" type="xs:integer"/>
  <xs:attribute name="nextOffset" use="optional" type="xs:nonNegativeInteger" />
  <xs:attribute name="maxCount" use="optional" type="xs:nonNegativeInteger" />
</xs:attributeGroup>
```

Figure 4. `dp:BasicPagingResponseAttributeGroup` Schema Fragment

3.2. Extended Pagination

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

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

3.2.1. Extended Pagination - Request attributes

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

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

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

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

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

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

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

```
<!-- ExtendedPagingAttributeGroup - adds set support  -->

<xs:attributeGroup name="ExtendedPagingAttributeGroup" >
  <xs:attributeGroup ref="BasicPagingAttributeGroup" />
  <xs:attribute name="setID"  use="optional" type="xs:string"/>
  <xs:attribute name="setReq" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Static"/>
        <xs:enumeration value="DeleteSet"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:attributeGroup>
```

Figure 5. `dp:ExtendedPagingAttributeGroup` Schema Fragment

3.2.2. Extended Pagination - Response Attributes

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

- `setID` **[Optional]** - the identity of the result set to which these results belong. This attribute MUST be specified when the results are associated with a static result set.
- `setExpires` **[Optional]** - the time at which this set will no longer be valid. This attribute MUST be specified in the initial response that creates the `setID` and MAY be specified in subsequent responses.

  The WSP is telling the WSC that it will maintain the static result set until this point in time and afterwards requests using that `setID` will fail.

```
<!-- ExtendedPagingResponseAttributeGroup - adds support for sets -->

<xs:attributeGroup name="ExtendedPagingResponseAttributeGroup">
  <xs:attributeGroup ref="BasicPagingResponseAttributeGroup" />
  <xs:attribute name="setID"     use="optional" type="xs:string"/>
  <xs:attribute name="setExpires" use="optional" type="xs:dateTime"/>
</xs:attributeGroup>
```

Figure 6. `dp:ExtendedPagingResponseAttributeGroup` Schema Fragment

3.3. Pagination Examples

3.3.1. Pagination Schema Example

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:ex:getdata"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:dp="urn:liberty:dp:2007-09"
 xmlns="urn:liberty:ex:getdata"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>
  <xs:import namespace="urn:liberty:dp:2007-09" schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>

  <xs:element name="GetData" type="GetDataType" />
  <xs:complexType name="GetDataType" mixed="true">
    <xs:attributeGroup ref="dp:ExtendedPagingAttributeGroup" />
  </xs:complexType>

  <xs:element name="GetDataResponse" type="GetDataResponseType" />
  <xs:complexType name="GetDataResponseType" mixed="true">
    <xs:attributeGroup ref="dp:ExtendedPagingResponseAttributeGroup" />
  </xs:complexType>

</xs:schema>
```

Example 5. GetData Schema incorporating pagination attributes

3.3.2. Basic Pagination Example

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

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

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

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

Example 6. Initial request, max of 10

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

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

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

Example 7. First response, remaining 11

3.3.2.3. 3. Request for 10 more items.

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

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

Example 8. Second request, max of 10

3.3.2.4. 4. Response to request with 1 remaining.

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

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

Example 9. Second response, remaining 1

3.3.2.5. 5. Request for 10 more items.

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

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

Example 10. Third request, max of 10

3.3.2.6. 6. Response to request with 0 remaining.

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

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

Example 11. Third response, remaining 0

3.3.3. Extended Pagination Example

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

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

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

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

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

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

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

```
<GetDataResponse xmlns="urn:liberty:ex:getdata" setID="38273923" setExpires="2007-01-14T17:33:11Z" remaining="11"

  ... data items go here (10 of them) ...
</GetDataResponse>
```

Example 13. First response, remaining 11, assigns setID

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

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

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

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

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

3.3.3.4. 4. Response to request with 1 remaining.

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

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

Example 15. Second response, remaining 1, uses setID

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

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

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

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

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

3.3.3.6. 6. Response to request with 0 remaining.

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

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

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

Example 17. Third response, remaining 0

3.4. Pagination Processing Rules

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

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

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

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

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

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

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

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

4. Delayed Notification

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

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

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

This design pattern solves the issues by providing:

1. a means for the invoker to use to indicate that they are able to receive delayed results and where to send such delayed (Section 4.2.1).

2. a means for immediate results to be returned, if available.

3. a means to indicate that immediate results are not available and will be returned later.

4. a means to deliver the delayed results to the invoker (including the definition of an interface that must be exposed by the invoker to receive such delayed results).

4.1. Delayed Notification Sequence

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

4.1.1. Step 1: Invoker submits request with `<dp:NotifyTo>` element

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

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

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

- The `<dp:NotifyTo>` element is required and the request fails if not specified.

- The service makes best efforts to verify that the future/indirect processing of the request will succeed and return the results of those efforts and accept that a later failure may occur without the ability to notify, directly, the invoker.

- The service waits for the completion of the process prior to returning the actual completion status and/or results to the invoker. This is more likely to be used in an indirect operation.

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

4.1.2. Step 2: Service instance performs initial processing

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

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

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

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

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

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

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

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

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

4.1.3. Step 3: The operation completes

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

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

4.2. Delayed Notification data structures

4.2.1. `<dp:NotifyTo>` Element

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

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

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

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

```
<!-- NotifyTo - element for carrying the notification destination -->

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

Figure 7. <dp:NotifyTo> — Schema Fragment

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

```
<!-- PMUpdate - update the PM for a existing PM at the ProvS -->

<xs:element name="PMUpdate" type="PMUpdateType"/>

<xs:complexType name="PMUpdateType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMUpdateItem"  maxOccurs="unbounded" />
        <xs:element ref="dp:NotifyTo"   minOccurs="0"      />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMUpdateItem" type="PMUpdateItemType" />

<xs:complexType name="PMUpdateItemType">
  <xs:sequence>
    <xs:element ref="PMDescriptor"/>
  </xs:sequence>
  <xs:attribute name="type"  type="xs:anyURI"  use="required"/>
  <xs:attribute name="itemID" type="xs:string"  use="required"/>
  <xs:attribute name="at"    type="xs:dateTime" use="optional"/>
</xs:complexType>
```

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

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

```
<prov:PMUpdate>
  <prov:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:778349-283920-88379-5448739</prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
      <ds:Signature>
        ... signature data goes here ...
      </ds:Signature>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <dp:NotifyTo>
    <wsa:Address>https://provider.com/notifications</wsa:Address>
    <wsa:Metadata>
      <ds:ProviderID>http://provider.com/</ds:ProviderID>
      <ds:ServiceType>urn:liberty:dp:2007-09:notification</ds:ServiceType>
      <ds:Framework version="2.0" />
      <ds:SecurityContext>
        <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:null</ds:SecurityMechID>
      </ds:SecurityContext>
    </wsa:Metadata>
  </dp:NotifyTo>
</prov:PMUpdate>
```

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

4.3. Delayed Notification Operations

4.3.1. Operation: *Notification*

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

4.3.1.1. `wsa:Action` values for `Notification` Messages

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

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

4.3.1.2. `Notification` Message

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

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

In other words, the two parties have switched roles.

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

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

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

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

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

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

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

```
<!-- Notification - interface for receiving the delayed completion status -->

<xs:element name="Notification" type="NotificationType"/>

<xs:complexType name="NotificationType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:any namespace="##other"
              processContents="lax"
              maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="ref" type="xs:string" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 8. <dp:Notification> — Schema Fragment

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

```
<dp:Notification ref="...messageID-of-request...">
  <prov:PMUpdateResponse>
    <lu:Status code="OK"/>
  </prov:PMUpdateResponse>
</dp:Notification>
```

Example 20. Example <dp:Notification> Message

4.3.1.3. NotificationResponse Message

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

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

```
<!-- NotificationResponse - the response to a Notification message -->

<xs:element name="NotificationResponse" type="NotificationResponseType" />

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

Figure 9. `<dp:NotificationResponse>` — Schema Fragment

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

```
<dp:NotificationResponse>
  <lu:Status code="OK" />
</dp:NotificationResponse>
```

Example 21. Example `<dp:NotificationResponse>` Message

4.3.1.4. Notification Processing Rules

- If the recipient is unable to locate a pending request with the message id specified in the `ref` attribute, the call MUST be treated as a failure. In such cases, if detailed status codes are being included, the detailed status code for this error MUST be *NotFound*.

- Each `<dp:Notification>` MUST ONLY include response data for a single request (the sender MAY NOT combine results from different requests).

- When building a notification message, if there are still outstanding operations for which the completion data or status is not available, the sender MUST indicate the current status of those items using the "*WillNotify*" status code.

- For multiple item requests that allow partial results, the sender MAY include the status of some or all of the pending items on a notification. So a single request with multiple request items could result in a single delayed notification, several delayed notifications, or even one delayed notification for each and every request item.

- When multiple item requests are supported, the same method used to match results in the non-delayed response MUST be used to match results in the delayed notification. Typically, the request items from the request would have an identifier that is placed to a `ref` attribute in the service level response item (note that this is different from the `ref` attribute in the `<dp:NotificationResponse>` element).

  On the other hand, if the normal service request uses order of elements in the response to match them to the request items, the sender MUST wait to accumulate the results for all items prior to returning any. This method is NOT RECOMMENDED.

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

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

4.4. Delayed Notification Examples (Non-Normative)

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

4.4.1. Step 1: Single request to update several PMDs

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

```
<prov:PMUpdate>
  <prov:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:cf5fab69-3092-4ef3-a7c0-f97e70ad769b<
/prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <prov:PMUpdateItem itemID="2" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:20643542-3f8c-4281-a4a6-
54b07d07ddab</prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <prov:PMUpdateItem itemID="3" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:ee059d84-8819-48c9-a46d-f7fbbc52866c</pro
v:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <prov:PMUpdateItem itemID="4" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:bf3cbd82-23c2-486e-b34b-301f
07827b61</prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <prov:PMUpdateItem itemID="5" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:7091b77c-b7e7-42d0-8fba-d6ddac45991e</prov:P
MID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <prov:PMUpdateItem itemID="6" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:e711fd21-10ba-41a9-bf46-5dfed4c0
441a</prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
  <dp:NotifyTo>
    <wsa:Address>https://provider.com/notifications</wsa:Address>
    <wsa:Metadata>
      <ds:ProviderID>http://provider.com/</ds:ProviderID>
      <ds:ServiceType>urn:liberty:dp:2007-09:notification</ds:ServiceType>
      <ds:Framework version="2.0" />
      <ds:SecurityContext>
        <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:null</ds:SecurityMechID>
      </ds:SecurityContext>
    </wsa:Metadata>
  </dp:NotifyTo>
</prov:PMUpdate>
```

Example 22. Step 1: request to update several PMDs

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

4.4.2. Step 2: Initial response from the Provisioning Service

Of the 6 requested items, the Provisioning service is able to process, immediately, one of them and one of them fails. The service returns the partial results shown below.

```
<prov:PMUpdateResponse>
  <lu:Status code="Partial">
    <lu:Status ref="1" code="WillNotify" />
    <lu:Status ref="2" code="OK" />
    <lu:Status ref="3" code="WillNotify" />
    <lu:Status ref="4" code="WillNotify" />
    <lu:Status ref="5" code="NotFound" />
    <lu:Status ref="6" code="WillNotify" />
  </lu:Status>
</prov:PMUpdateResponse>
```

Example 23. Step 2: Initial response

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

```
<prov:PMUpdateResponse>
  <lu:Status code="WillNotify" />
</prov:PMUpdateResponse>
```

Example 24. Initial response with no completed operations

4.4.3. Step 3: First Notification

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

```
<dp:Notification ref="uuid:2376bf6d-9cf6-420e-a438-436a30f7d3f1" >
  <prov:PMUpdateResponse>
    <lu:Status code="Notify">
      <lu:Status ref="1" code="WillNotify" />
      <lu:Status ref="3" code="OK" />
      <lu:Status ref="4" code="WillNotify" />
      <lu:Status ref="6" code="WillNotify" />
    </lu:Status>
  </prov:PMUpdateResponse>
</dp:Notification>
```

Example 25. Step 3: First Notification

4.4.4. Step 4: Second Notification

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

```
<dp:Notification ref="uuid:2376bf6d-9cf6-420e-a438-436a30f7d3f1" >
  <prov:PMUpdateResponse>
    <lu:Status code="Notify">
      <lu:Status ref="1" code="OK" />
      <lu:Status ref="4" code="Failed" />
      <lu:Status ref="6" code="WillNotify" />
    </lu:Status>
  </prov:PMUpdateResponse>
</dp:Notification>
```

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**

5. ID-WSF Design Patterns Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:dp:2007-09"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns="urn:liberty:dp:2007-09"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>

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

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

  <!--*********************************************************-->
  <!--                                                       -->
  <!--       Polling design pattern schema defs              -->
  <!--                                                       -->
  <!--*********************************************************-->


  <!-- PollType - datatype for polling the recipient for any new work -->

  <xs:complexType name="PollType">
    <xs:sequence>
      <xs:element ref="wsa:Action" minOccurs="0"  maxOccurs="unbounded" />
      <xs:element ref="Response"   minOccurs="0"  maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="wait" type="xs:integer" use="required" />
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

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


  <!-- PollResponseType - the datatype of response to a polling message -->

  <xs:complexType name="PollResponseType">
    <xs:complexContent>
      <xs:extension base="ResponseAbstractType">
        <xs:sequence>
          <xs:element ref="Request" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="nextPoll" type="xs:integer" use="optional" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Request" type="RequestType" />
  <xs:complexType name="RequestType">
    <xs:sequence>
      <xs:any namespace="##other"
            processContents="lax"
            minOccurs="0"
```

```
          maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>


<!--*************************************************************-->
<!--                                                             -->
<!--    Pagination of results design pattern schema defs     -->
<!--                                                             -->
<!--*************************************************************-->


<!--BasicPagingAttributeGroup - basic request pagination support -->

<xs:attributeGroup name="BasicPagingAttributeGroup" >
  <xs:attribute name="count" use="optional"  type="xs:nonNegativeInteger"/>
  <xs:attribute name="offset" use="optional" type="xs:nonNegativeInteger" />
</xs:attributeGroup>



<!-- BasicPagingResponseAttributeGroup - basic response pagination support -->

<xs:attributeGroup name="BasicPagingResponseAttributeGroup">
  <xs:attribute name="remaining" use="optional" type="xs:integer"/>
  <xs:attribute name="nextOffset" use="optional" type="xs:nonNegativeInteger" />
  <xs:attribute name="maxCount" use="optional" type="xs:nonNegativeInteger" />
</xs:attributeGroup>


<!-- ExtendedPagingAttributeGroup - adds set support  -->

<xs:attributeGroup name="ExtendedPagingAttributeGroup" >
  <xs:attributeGroup ref="BasicPagingAttributeGroup" />
  <xs:attribute name="setID"  use="optional" type="xs:string"/>
  <xs:attribute name="setReq" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Static"/>
        <xs:enumeration value="DeleteSet"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:attributeGroup>



<!-- ExtendedPagingResponseAttributeGroup - adds support for sets -->

<xs:attributeGroup name="ExtendedPagingResponseAttributeGroup">
  <xs:attributeGroup ref="BasicPagingResponseAttributeGroup" />
  <xs:attribute name="setID"    use="optional" type="xs:string"/>
  <xs:attribute name="setExpires" use="optional" type="xs:dateTime"/>
</xs:attributeGroup>


<!--*************************************************************-->
<!--                                                             -->
<!--    Delayed Notification design pattern schema defs      -->
<!--                                                             -->
<!--*************************************************************-->


<!-- NotifyTo - element for carrying the notification destination -->

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

```
  <!-- Notification - interface for receiving the delayed completion status -->

  <xs:element name="Notification" type="NotificationType"/>

  <xs:complexType name="NotificationType">
    <xs:complexContent>
      <xs:extension base="RequestAbstractType">
        <xs:sequence>
          <xs:any namespace="##other"
                processContents="lax"
                maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="ref" type="xs:string" use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- NotificationResponse - the response to a Notification message -->

  <xs:element name="NotificationResponse" type="NotificationResponseType" />

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


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

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


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

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


</xs:schema>
```

6. Notification Endpoint WSDL (non-normative)

```
<?xml version="1.0"?>
<definitions name="notify-svc"
 targetNamespace="urn:liberty:notify:2007-09"
 xmlns:tns="urn:liberty:notify:2007-09"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
 xmlns:dp="urn:liberty:dp:2007-09"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
                http://schemas.xmlsoap.org/wsdl/
    http://www.w3.org/2006/02/addressing/wsdl
    http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

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

 <message name="Notification">
  <part name="body" element="dp:Notification"/>
 </message>
 <message name="NotificationResponse">
  <part name="body" element="dp:NotificationResponse"/>
 </message>

 <portType name="NotifyPort">

  <operation name="Notification">
   <input message="tns:Notification"
     wsaw:Action="urn:liberty:dp:2007-09:Notification" />
   <output message="tns:NotificationResponse"
    wsaw:Action="urn:liberty:dp:2007-09:NotificationResponse" />
  </operation>

 </portType>

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

 <binding name="NotifyBinding" type="tns:NotifyPort">

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

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

 </binding>

 <service name="NotifyService">

  <port name="NotifyPort" binding="tns:NotifyBinding">

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

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

```
    </port>

  </service>

</definitions>
```

References

Normative

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

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

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

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

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

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

[LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. " Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification ," v2.0-errata-1.0-01, Liberty Alliance Project (28 November, 2006). *http://www.projectliberty.org/specs*

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

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

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

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

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

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

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

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

SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf*

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

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

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

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

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

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

Informative

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

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

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