



Liberty ID-WSF Provisioning Service Specification

Version: 1.0-02

Editors:

Conor P. Cahill, Intel Corporation

Contributors:

Hiroyoshi Takiguchi, NTT

Hubert Le Van Gong, Sun

Paul Madsen, NTT

Greg Whitehead, HP

Abstract:

This specification defines the Provisioning Service interfaces.

Filename: liberty-idwsf-prov-v1.0-02.pdf

1

Notice

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

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

15 Copyright © 2006 2FA Technology; Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.;
16 American Express Company; Amssoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Bank of
17 America Corporation; Beta Systems Software AG; British Telecommunications plc; Computer Associates
18 International, Inc.; Credentica; DataPower Technology, Inc.; Deutsche Telekom AG, T-Com; Diamelle Technologies,
19 Inc.; Diversinet Corp.; Drummond Group Inc.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Falkin
20 Systems LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
21 développement de l'administration électronique (ADAE); Fugen Solutions, Inc; Fulvens Ltd.; GSA Office of
22 Governmentwide Policy; Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke & Devrient GmbH;
23 Hewlett-Packard Company; Hochhauser & Co., LLC; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega;
24 Kayak Interactive; Livo Technologies; Luminance Consulting Services; MasterCard International; MedCommons
25 Inc.; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; NTT DoCoMo, Inc.; Netegrity, Inc.; Neustar, Inc.;
26 New Zealand Government State Services Commission; Nippon Telegraph and Telephone Corporation; Nokia
27 Corporation; Novell, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation; RSA Security Inc.;
28 Reactivity Inc.; Royal Mail Group plc; SAP AG; Senforce; Sharp Laboratories of America; Sigaba; SmartTrust; Sony
29 Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.; Telecom Italia S.p.A.;
30 Telefónica Móviles, S.A.; Telenor R&D; Thales e-Security; Trusted Network Technologies; UNINETT AS; UTI;
31 VeriSign, Inc.; Vodafone Group Plc.; Wave Systems Corp. All rights reserved.

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

38 Contents

39	1. Introduction	5
40	1.1. Notation and Conventions	5
41	1.1.1. XML Namespaces	5
42	2. Overview	6
43	2.1. Provisioning Components	6
44	3. Data Definitions	8
45	3.1. Provisioned Module Identifier	8
46	3.2. ProvisioningHandle	8
47	3.3. PMDescriptor	9
48	3.4. PMStatus	11
49	3.5. Callback EPR	13
50	4. Provisioning Service (ProvS)	14
51	4.1. Service URIs	14
52	4.2. Status Codes	14
53	4.3. Request and Response Abstract Types	15
54	4.3.1. Complex Type RequestAbstractType	15
55	4.3.2. Complex Type ResponseAbstractType	15
56	4.4. Operation: <i>GetPMEngine</i>	15
57	4.4.1. wsa:Action values for GetPMEngine Messages	15
58	4.4.2. GetPMEngine Message	16
59	4.4.3. GetPMEngineResponse Message	17
60	4.4.4. GetPMEngine Processing Rules	18
61	4.5. Operation: <i>PMArtifactResolve</i>	18
62	4.5.1. wsa:Action values for PMArtifactResolve Messages	18
63	4.5.2. PMArtifactResolve Message	18
64	4.5.3. PMArtifactResolveResponse Message	19
65	4.5.4. PMArtifactResolve Processing Rules	20
66	4.6. Operation: <i>PMDActivate</i>	21
67	4.6.1. wsa:Action values for PMDActivate Messages	21
68	4.6.2. PMDActivate Message	21
69	4.6.3. PMDActivateResponse Message	22
70	4.6.4. PMDActivate Processing Rules	23
71	4.7. Operation: <i>PMDDeactivate</i>	23
72	4.7.1. wsa:Action values for PMDDeactivate Messages	23
73	4.7.2. PMDDeactivate Message	23
74	4.7.3. PMDDeactivateResponse Message	25
75	4.7.4. PMDDeactivate Processing Rules	25
76	4.8. Operation: <i>PMDDelete</i>	26
77	4.8.1. wsa:Action values for PMDDelete Messages	26
78	4.8.2. PMDDelete Message	26
79	4.8.3. PMDDeleteResponse Message	27
80	4.8.4. PMDDelete Processing Rules	27
81	4.9. Operation: <i>PMDGetStatus</i>	27
82	4.9.1. wsa:Action values for PMDGetStatus Messages	28
83	4.9.2. PMDGetStatus Message	28
84	4.9.3. PMDGetStatusResponse Message	28
85	4.9.4. PMDGetStatus Processing Rules	29
86	4.10. Operation: <i>PMDRegister</i>	30
87	4.10.1. wsa:Action values for PMDRegister Messages	30
88	4.10.2. PMDRegister Message	30
89	4.10.3. PMDRegisterResponse Message	31
90	4.10.4. PMDRegister Processing Rules	32

91	4.11. Operation: <i>PMDSetStatus</i>	32
92	4.11.1. <i>wsa:Action</i> values for <i>PMDSetStatus</i> Messages	32
93	4.11.2. <i>PMDSetStatus</i> Message	33
94	4.11.3. <i>PMDSetStatusResponse</i> Message	33
95	4.11.4. <i>PMDSetStatus</i> Processing Rules	34
96	4.12. Operation: <i>PMDUpdate</i>	34
97	4.12.1. <i>wsa:Action</i> values for <i>PMDUpdate</i> Messages	34
98	4.12.2. <i>PMDUpdate</i> Message	34
99	4.12.3. <i>PMDUpdateResponse</i> Message	36
100	4.12.4. <i>PMDUpdate</i> Processing Rules	37
101	4.13. Operation: <i>Poll</i>	37
102	4.13.1. <i>wsa:Action</i> values for <i>Poll</i> Messages	37
103	4.13.2. <i>Poll</i> Message	37
104	4.13.3. <i>PollResponse</i> Message	38
105	4.13.4. <i>Poll</i> Processing Rules	39
106	4.14. Operation: <i>UpdateEPR</i>	39
107	4.14.1. <i>wsa:Action</i> values for <i>UpdateEPR</i> Messages	39
108	4.14.2. <i>UpdateEPR</i> Message	40
109	4.14.3. <i>UpdateEPRResponse</i> Message	41
110	4.14.4. <i>UpdateEPR</i> Processing Rules	41
111	5. Security and Privacy Considerations	43
112	A. Provisioning Service Schema	44
113	B. Provisioning Service WSDL	52
114	References	57

115 **1. Introduction**

116 Provisioning, in this context, refers to the distribution, installation and maintenance (update/delete) of some functional
117 module (perhaps a TM) onto a device or platform. The specific capabilities and features of a particular functional
118 module are out of scope here. This process is only concerned with getting the functional module up and running
119 within the target environment.

120 **1.1. Notation and Conventions**

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

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

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

128 **1.1.1. XML Namespaces**

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

130 • The prefix *prov*: represents the Provisioning Service namespace. This namespace is the default for instance
131 fragments, type names, and element names in this document. In schema listings, and in examples of service
132 messages and fragments thereof, this is the default namespace *when* no prefix is shown:

133 *urn:liberty:prov:2006-12*

134 • The prefix *pmm*: stands for the Liberty ID-WSF Provisioned Module Manager namespace [\[LibertyPMM\]](#):

135 *urn:liberty:pmm:2006-12*

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

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

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

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

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

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

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

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

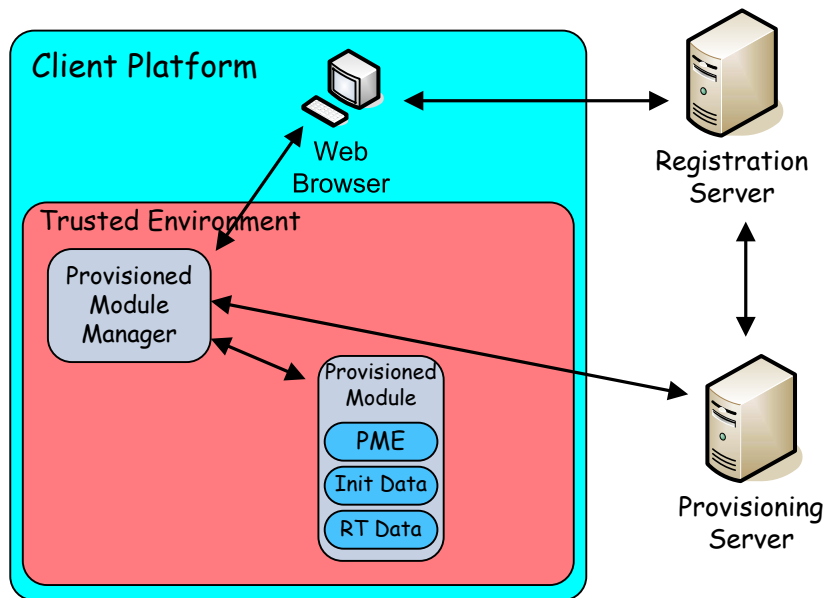
144 2. Overview

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

148 This document describes the Liberty ID-WSF Provisioning Service (ProvS). The ProvS is the entity which distributes
149 the data and potentially executable code, to the client platforms. The ProvS also provides a control point for lifecycle
150 management of provisioned modules (PMs), supporting operations such as update, delete, activate and deactivate.

151 2.1. Provisioning Components

152 The following diagram illustrates the components involved in the provisioning process:



153

154 **Figure 1. Provisioning Components**

155 Things to note about this diagram:

- 156 • It is not drawn to any form of scale!
- 157 • The client platform represents any type of client, such as a personal computer, a device, a smart card, etc..
- 158 • The trusted environment represents some form of tamper resistant container (thus providing a level of trust for
159 the provisioned components). The trusted environment is **not** a requirement of these protocols – the components
160 shown within the trusted environment could very well exist directly within the relatively untrusted client platform
161 (e.g. the Provisioned Module Manager could run as a service within the Client Platform operating system).
- 162 • The Provisioned Module Manager (PMM) is a service running on the client platform which provides a beach
163 head for provisioning operations. The PMM exposes the interfaces documented within the Liberty ID-WSF
164 Provisioned Module Manager Service Specification [[LibertyPMM](#)].

165 This document does not address the chicken-vs-egg issue of how the PMM comes into being on the client. It may
166 be built into the platform or it may be manually installed by some party (such as the user). That discussion is
167 out-of-scope.

- 168 • The Provisioned Module (PM) is a component which performs some set of functionality. For example, a PM
169 could be a TM (a module which provides IdP extension functionality). PMs may also expose functionality that is
170 not defined by Liberty specifications.
- 171 The Provisioned module is shown as being composed of 3 distinct parts:
- 172 • **Provisioned Module Engine (PME)** - the executable code which provides the functionality for the PM.
173 This is defined as a separate component here to enable a provisioning process which allows the PME to pre-
174 exist in the client platform and so just delivers the data necessary to instantiate the PM using that pre-existing
175 engine. Of course, the PME may not pre-exist and in such cases the PMM will have to retrieve it.
- 176 During provisioning, the PME is passed by reference (name) so that the PMM can determine whether or not
177 the PME already exists (either because it was pre-installed or because the same PME has been previously
178 provisioned). Should the PMM need to obtain the PME, the passed in reference is used to identify the PME
179 being downloaded.
- 180 • **Initialization Data (PMInitData)** - the data needed by the PME in order to initialize a new instance of a
181 PM. This may be the actual data needed by the PME or it may be a reference that the PME knows how
182 to dereference and obtain the initialization data at runtime. This data may or may not be needed during the
183 provisioning process. Some PMs are fully individualized and have their PMInitData built in.
- 184 The format and structure of the PMInitData is out of scope for this document and is specific to the PME. It is
185 up to the Provisioning Service to resolve what data is needed for what PME. The PMM treats PMInitData as
186 an opaque data set that it passes to the PME upon initialization.
- 187 • **Runtime Data (PMRTData)** - the runtime data created/managed by the PM instance as it performs its tasks.
188 This would include things like MINGs for a TM that is minting assertions, private keys, etc. This is defined
189 separate from the InitData to allow for PM portability (where a previously activated PM is moved to another
190 client platform).
- 191 • The Web Browser in this diagram represents an enhanced browser (either directly or via a plug-in) with support for
192 the provisioning process. In other provisioning use cases this may be an application or can even be the PMM
193 itself can instigate a new provision operation (typically via some direct interaction with the user).
- 194 • The Registration Server (RS) is not a Liberty defined entity, but rather a deployment component for a particular
195 set of use cases. In this use case, the RS interacts with the user through a web browser and then controls the
196 provisioning process using the interfaces on the Provisioning Server.
- 197 • The Provisioning Server (ProvS) is typically a network hosted service that is the primary entity with which the
198 PMM interacts. This server is an instance of a Liberty ID-WSF Provisioning Service (see [[LibertyPROV](#)]).
199 The primary function of the ProvS is to provide a trusted endpoint for the management and distribution of PMs.

200 3. Data Definitions

201 3.1. Provisioned Module Identifier

202 A PMID is a unique identifier assigned to the PM at the point the PMD is first registered with the ProvS (i.e. before
203 the PM has been provisioned). This identifier is used on all subsequent communications with the ProvS relating to
204 that PMD from the registering party as well as from the PMM. Therefore the value selected for the PMID MUST be
205 unique across all such values issued by that ProvS.

206 The <PMID> element carries the identifier value as its content and has no other sub-elements. The following attributes
207 are defined:

- 208 • **issuer [Required]** - the Provider ID of the ProvS which issued the PMID. This is used to prevent collision of
209 PMID values at the PMM where multiple ProvS entities may be provisioning PMs.

210 The following schema fragment defines the the **PMID** element:

```
211 <!-- PMID - the Provisioned Module Identifier -->  
212  
213 <xs:element name="PMID" type="PMIDType" />  
214 <xs:complexType name="PMIDType">  
215 <xs:attribute name="issuer" type="xs:anyURI" use="required" />  
216 </xs:complexType>  
217
```

218 The following is an example of a **PMID** element.

```
219 <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230 328-92379-2397923</prov:PMID>  
220
```

221 3.2. ProvisioningHandle

222 A Provisioning Handle provides the PMM with the necessary information to invoke the ProvS and an artifact which
223 refers to a specific PM that is to be provisioned. This is typically handed to the PMM through some non-liberty
224 protocols (such as distribution via email or by being downloaded by the principal).

225 The <ProvisioningHandle> element has the following elements/attributes:

- 226 • <PMArtifact> **[Required]** - a token issued by the ProvS which identifies a specific PM that is to be provisioned
227 to the Advanced client.

228 The contents and structure of the PMArtifact are not defined by this specification and should be treated as an
229 opaque blob by the PMM and any intermediary handlers.

230 Any entity that may pass along or process the PH MUST support a <PMArtifact> length of 2048 bytes. The
231 ProvS, when creating the <PMArtifact> MUST ensure it fits within 2048 bytes.

232 The ProvS should ensure that only appropriate PMMs are able to present the PMArtifact and it can only be done
233 once (in order to ensure that the Provisioning Handle hasn't been intercepted and used by another party without at
234 least recognizing this when the intended party tries to use it).

- 235 • <ProvisioningServiceEPR> **[optional]** - an ID-WSF EPR describing the endpoint for the entity that is
236 provisioning this PM.

237 Normally the EPR is specified within the PH. However, there MAY be some environments where the PMM has
238 pre-existing knowledge of the ProvS EPR and so this element is listed as optional to allow it to be not specified in
239 such cases.

- 240 • `<ds:Signature>` [**Optional**] - an XML digital signature ([XMLDsig]) covering the entire descriptor.
- 241 If a signature is present, the signer **MUST** follow the normative rules laid out in Section 5 ("SAML and XML
- 242 Signature Syntax and Processing") of the SAML 2.0 Core Specification ([SAMLCore2]).
- 243 • `anyAttribute` [**Optional**] - zero or more attributes from a namespace other than that of this specification. One
- 244 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

245 The following schema fragment defines the the **ProvisioningHandle** element:

```

246 <!-- ProvisioningHandle - Info for PMM to initiate provisioning process -->
247
248 <xs:element name="ProvisioningHandle" type="ProvisioningHandleType" />
249 <xs:complexType name="ProvisioningHandleType">
250   <xs:sequence>
251     <xs:element ref="PMArtifact" />
252     <xs:element ref="ProvisioningServiceEPR" minOccurs="0" />
253     <xs:element ref="ds:Signature" minOccurs="0" />
254   </xs:sequence>
255   <xs:anyAttribute namespace="##other" processContents="lax" />
256 </xs:complexType>
257
258 <xs:element name="ProvisioningServiceEPR" type="wsa:EndpointReferenceType" />
259
260 <xs:element name="PMArtifact" type="xs:string" />
261

```

262 The following is an example of a **ProvisioningHandle** element.

```

263 <prov:ProvisioningHandle xs:id="2302384823023">
264   <prov:PMArtifact>23asdfhoi323hposdf92 3h9sdfhwiorh2398asdfjweoiha</prov:PMArtifact>
265   <prov:ProvisioningServiceEPR>
266     <wsa:Address>http://provision.idpsRus.com</wsa:Address>
267     <wsa:Metadata>
268       <ds:Abstract>Provisioning Service</ds:Abstract>
269       <ds:ProviderID>http://provisioning-provider.idpsRus.com/</ds:ProviderID>
270       <ds:ServiceType>urn:liberty:prov:2006-12</ds:ServiceType>
271       <ds:Framework version="2.0" />
272       <ds:SecurityContext>
273         <ds:SecurityMechID>
274           urn:liberty:security:2005-02:TLS:SAMLV2
275         </ds:SecurityMechID>
276         <sec:Token ref="urn:liberty:disco:tokenref:ObtainFromIDP" />
277       </ds:SecurityContext>
278     </wsa:Metadata>
279   </prov:ProvisioningServiceEPR>
280   <ds:Signature>
281     ... signature info here ..
282   </ds:Signature>
283 </prov:ProvisioningHandle>
284

```

285 3.3. PMDescriptor

286 The `<PMDescriptor >`, or Provisioned Module Descriptor (PMD), contains the information necessary for the PMM

287 to instantiate a PM within the Advanced Client.

288 The `<PMDescriptor >` is typically acquired by the PMM following the dereference of the `<PMArtifact>` at the

289 ProvS.

290 The `<PMDescriptor >` element has the following elements/attributes:

- 291 • <PMID> **[Required]** - the Provisioned Module IDentifier for the PM described by this descriptor.
- 292 • <PMEngineRef> **[Optional]** - a URI reference to a PMEngine. This engine may already exist on the
293 Advanced Client, or it may need to be downloaded and installed by the PMM. This element **MUST** be present
294 in any <PMDescriptor> that is describing a complete PM. However, the element **MAY** be absent when the
295 <PMDescriptor> is used to describe an update.
- 296 • <PMInitData> **[Optional]** - initialization data needed by the PMEngine to initialize as a PM instance. This is
297 an encrypted data chunk which typically only the PMEngine can decrypt.
- 298 • <PMRTData> **[Optional]** - runtime data saved by a previous instance of the PM associated with this descriptor.
299 The PM will usually process this after it has been initialized. This is also an encrypted data chunk which typically
300 only the PMEngine can decrypt.
- 301 • <ds:Signature> **[Optional]** - an XML digital signature ([\[XMLDsig\]](#)) covering the entire descriptor.
302 If a signature is present, the signer **MUST** follow the normative rules laid out in Section 5 ("SAML and XML
303 Signature Syntax and Processing") of the SAML 2.0 Core Specification ([\[SAMLCore2\]](#)).
- 304 • activate **[optional]** - a boolean attribute indicating whether the PM should be activated upon installation. If
305 this attribute is not specified or its value is *true*, the PM is activated. Otherwise the PM is installed, but not
306 activated.
- 307 See the related activateAt attribute below for automated future activation.
- 308 • activateAt **[optional]** - an attribute indicating when the PM should be activated. The value of this attribute is
309 a specific instance in time at which the PM should be attributed.
- 310 This is typically used when a PM is to be installed now for a future activation.
- 311 • anyAttribute **[Optional]** - zero or more attributes from a namespace other than that of this specification. One
312 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

313 The following schema fragment defines the **PMDescriptor** element:

```

314 <!-- PMDescriptor - describes/carries the components of a PM -->
315
316 <xs:element name="PMDescriptor" type="PMDescriptorType" />
317
318 <xs:complexType name="PMDescriptorType">
319   <xs:sequence>
320     <xs:element ref="PMID" />
321     <xs:element ref="PMEngineRef" minOccurs="0" />
322     <xs:element ref="PMInitData" minOccurs="0" />
323     <xs:element ref="PMRTData" minOccurs="0" />
324     <xs:element ref="ds:Signature" minOccurs="0" />
325   </xs:sequence>
326   <xs:attribute name="activate" type="xs:boolean" use="optional" />
327   <xs:attribute name="activateAt" type="xs:dateTime" use="optional" />
328   <xs:anyAttribute namespace="##other" processContents="lax" />
329 </xs:complexType>
330
331 <xs:element name="PMEngineRef" type="xs:anyURI" />
332 <xs:element name="PMInitData" type="EncryptedElementType" />
333 <xs:element name="PMRTData" type="EncryptedElementType" />
334
335 <xs:complexType name="EncryptedElementType">
336   <xs:sequence>
337     <xs:element ref="xenc:EncryptedData"/>
338     <xs:element ref="xenc:EncryptedKey" minOccurs="0" maxOccurs="unbounded"/>
339   </xs:sequence>
340 </xs:complexType>
341

```

342 The following is an example of a **PMDescriptor** element. In this case, the descriptor is signed, includes the reference
 343 to a PM Engine as well as some initialization data, and the PM is activated upon installation.

```
344 <prov:PMDescriptor xs:id="2323923900239" >
345   <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
346   <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
347   <prov:PMInitData>
348     <xenc:EncryptedData>
349       .... encrypted data here ...
350     </xenc:EncryptedData>
351   </prov:PMInitData>
352   <ds:Signature>
353     ... signature data goes here ...
354   </ds:Signature>
355 </prov:PMDescriptor>
356
```

357 The following is another example of a **PMDescriptor** element. In this case, the PM has no initialization data (which
 358 would typically mean the PMEngine is somehow uniquely branded with the data) and it is activated at some point in
 359 the future.

```
360 <prov:PMDescriptor xs:id="2323923900239"
361   activateAt="2006-12-11T14:52:00Z" >
362   <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
363   <prov:PMEngineRef>http://pmsRus.org/ConorsPM/1.0</prov:PMEngineRef>
364   <ds:Signature>
365     ... signature data goes here ...
366   </ds:Signature>
367 </prov:PMDescriptor>
368
```

369 The following is another example of a **PMDescriptor** element. In this case, the PM is **not** activated when it is
 370 installed. In order to activate it, the provisioning service must invoke the PMM `<pmm:PMActivate` interface.

```
371 <prov:PMDescriptor xs:id="2323923900239"
372   activate="false" >
373   <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
374   <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
375   <prov:PMInitData>
376     <xenc:EncryptedData>
377       .... encrypted data here ...
378     </xenc:EncryptedData>
379   </prov:PMInitData>
380   <ds:Signature>
381     ... signature data goes here ...
382   </ds:Signature>
383 </prov:PMDescriptor>
384
```

385 3.4. PMStatus

386 The `<PMStatus >`, (or Provisioned Module Status) describes the provisioning status of a particular provisioned
 387 module.

388 The `<PMStatus >` element has the following attributes:

- 389 • `<PMID>` **[Required]** - the PMID for the PM that this status applies to.
- 390 • `<State>` **[Required]** - the current state of the provisioning of this PM. This element contains the URI value
 391 representing the current state as well as the following attribute:

392 • *asof* [**Optional**] - the times when the current status of the PM was achieved. This attribute SHOULD NOT
393 be specified when the status is *urn:liberty:prov:2006-12:status:NotFound*. Otherwise it should always be
394 present.

395 The following status values are defined by Liberty for the provisioning status of a PM (implementations MAY define
396 additional values and attach specific meaning to them):

397 • *urn:liberty:prov:2006-12:status:Registered* - the *<PMDescriptor>* has been registered with the ProvS and the
398 ProvS is awaiting the *<PMArtifactResolve>* request from the PMM.

399 • *urn:liberty:prov:2006-12:status:Resolved* - the PMM has obtained the *<PMDescriptor>* via the
400 *<prov:PMArtifactResolve>* request and the ProvS is awaiting a status update from the PMM to indi-
401 cate the provisioning is complete.

402 • *urn:liberty:prov:2006-12:status:Activated* - the PM is installed and active.

403 • *urn:liberty:prov:2006-12:status:ActivateQueued* - an activation request has been received at the ProvS, but not
404 yet transmitted to the PMM.

405 • *urn:liberty:prov:2006-12:status:Activating* - an activation request has been sent to the PMM, but the processing
406 at the PMM is not yet complete.

407 • *urn:liberty:prov:2006-12:status:InitFailed* - the PMM failed to initialize the PM.

408 • *urn:liberty:prov:2006-12:status:Deactivated* - the PM is installed, but is currently inactive

409 • *urn:liberty:prov:2006-12:status:DeactivateQueued* - a deactivation request has been received at the ProvS, but
410 not yet transmitted to the PMM.

411 • *urn:liberty:prov:2006-12:status:Deactivating* - a deactivation request has been transmitted to the PMM, but the
412 processing at the PMM is not yet complete.

413 • *urn:liberty:prov:2006-12:status:Deleted* - the PM has been deleted.

414 • *urn:liberty:prov:2006-12:status>DeleteQueued* - a delete has been queued, but has not been initiated at the PMM
415 yet.

416 • *urn:liberty:prov:2006-12:status:Deleting* - a delete has been initiated at the PMM, but is not complete.

417 • *urn:liberty:prov:2006-12:status:UpdateQueued* - an update has been requested for the PM, but has not yet been
418 provided to the PMM (most likely because the ProvS is awaiting a *<prov:Poll>* request from the PMM).

419 • *urn:liberty:prov:2006-12:status:Updating* - an update request has been made to the PMM and the ProvS is
420 awaiting a status update from the PMM to indicate that the update is complete. Once the update is complete, the
421 status will return to *urn:liberty:prov:2006-12:status:Activated* or *urn:liberty:prov:2006-12:status:Deactivated*
422 (whichever the PM was in prior to the update).

423 The following schema fragment defines the **PMStatus** element:

```

424 <!-- PMStatus - Provisioning status of the PM -->
425
426 <xs:element name="PMStatus" type="PMStatusType" />
427 <xs:complexType name="PMStatusType">
428   <xs:sequence>
429     <xs:element ref="PMID" />
430     <xs:element ref="State" />
431   </xs:sequence>
432 </xs:complexType>
433
434 <xs:element name="State" type="StateType" />
435 <xs:complexType name="StateType">
436   <xs:simpleContent>
437     <xs:extension base="xs:anyURI">
438       <xs:attribute name="asof" type="xs:dateTime" use="optional" />
439     </xs:extension>
440   </xs:simpleContent>
441 </xs:complexType>
442
```

443 The following is an example of a **PMStatus** element. In this case, the provisioning of the PM is complete.

```

444 <prov:PMStatus>
445   <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2 397923</prov:PMID>
446   <prov:State asof="2006-12-14T17:31:11Z">urn:liberty:prov:2006-12:status:Activated</prov:State>
447 </prov:PMStatus>
448
```

449 3.5. Callback EPR

450 The `<prov:CallbackEPR>` is used by the PMM to register its callback location for PM maintenance operations.
 451 This EPR is normally a traditional ID-WSF EPR (see [\[LibertyDisco\]](#)).

452 However, in the case where the PMM cannot expose an external endpoint that is visible to the ProvS, the PMM should
 453 register an "anonymous" `<prov:CallbackEPR>` which **MUST** have the following characteristics:

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

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

```

457 <!-- CallbackEPR - where the PMM can receive Provisionig update requests -->
458
459 <xs:element name="CallbackEPR" type="wsa:EndpointReferenceType"/>
460
```

461 **Figure 2. `<prov:CallbackEPR>` — Schema Fragment**

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

```

463 <prov:CallbackEPR>
464   <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
465 </prov:CallbackEPR>
466
```

467 **Example 1. Example anonymous `<prov:CallbackEPR>`**

468 4. Provisioning Service (ProvS)

469 The Provisioning Service (ProvS) provides the interfaces used by PMMs on Advanced Clients to obtain PMs for
 470 provisioning.

471 An abstract WSDL definition for the Provisioning Service is included in this document, see [Appendix B: Provisioning](#)
 472 [Service WSDL](#) . This WSDL document defines all of the "WSDL operations" for the IdP Service.

473 The complete schema for the Provisioning Service is included in this document, see [Appendix A: Provisioning Service](#)
 474 [Schema](#) .

475 4.1. Service URIs

476 **Table 1. ProvS Service URIs**

Use	URI
Service Type	<i>urn:liberty:prov:2006-12</i>
GetPMEngine wsa:Action	<i>urn:liberty:prov:2006-12:GetPMEngine</i>
GetPMEngineResponse wsa:Action	<i>urn:liberty:prov:2006-12:GetPMEngineResponse</i>
PMArtifactResolve wsa:Action	<i>urn:liberty:prov:2006-12:PMArtifactResolve</i>
PMArtifactResolveResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMArtifactResolveResponse</i>
PMDActivate wsa:Action	<i>urn:liberty:prov:2006-12:PMDActivate</i>
PMDActivateResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDActivateResponse</i>
PMDDeactivate wsa:Action	<i>urn:liberty:prov:2006-12:PMDDeactivate</i>
PMDDeactivateResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDDeactivateResponse</i>
PMDDDelete wsa:Action	<i>urn:liberty:prov:2006-12:PMDDDelete</i>
PMDDDeleteResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDDDeleteResponse</i>
PMDGetStatus wsa:Action	<i>urn:liberty:prov:2006-12:PMDGetStatus</i>
PMDGetStatusResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDGetStatusResponse</i>
PMDRegister wsa:Action	<i>urn:liberty:prov:2006-12:PMDRegister</i>
PMDRegisterResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDRegisterResponse</i>
PMDSetStatus wsa:Action	<i>urn:liberty:prov:2006-12:PMDSetStatus</i>
PMDSetStatusResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDSetStatusResponse</i>
PMDUpdate wsa:Action	<i>urn:liberty:prov:2006-12:PMDUpdate</i>
PMDUpdateResponse wsa:Action	<i>urn:liberty:prov:2006-12:PMDUpdateResponse</i>
Poll wsa:Action	<i>urn:liberty:prov:2006-12:Poll</i>
PollResponse wsa:Action	<i>urn:liberty:prov:2006-12:PollResponse</i>
UpdateEPR wsa:Action	<i>urn:liberty:prov:2006-12:UpdateEPR</i>
UpdateEPRResponse wsa:Action	<i>urn:liberty:prov:2006-12:UpdateEPRResponse</i>

477 **4.2. Status Codes**

478 The following status code strings are defined:

479 • *OK*: message processing succeeded

480 • *Failed*: general failure code

481 These strings are expected to appear in the "code" attribute of `<Status>` elements used in SOAP-bound Provisioning
482 Service protocol messages. Specific uses for the status codes are defined in the processing rules for individual
483 messages. The contents of the `comment` attribute are not defined by this specification, but it may be used for additional
484 descriptive text intended for human consumption (for example, to carry information that will aid debugging).

485 **4.3. Request and Response Abstract Types**

486 **4.3.1. Complex Type RequestAbstractType**

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

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

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

```
492 <!-- RequestAbstractType - common request message structure -->  
493  
494 <xs:complexType name="RequestAbstractType" abstract="true">  
495   <xs:anyAttribute namespace="##other" processContents="lax"/>  
496 </xs:complexType>  
497
```

498 **4.3.2. Complex Type ResponseAbstractType**

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

501 `<lu:Status>` [**Required**] The `<lu:Status>` element is used to convey status codes and related information. The
502 schema fragment is defined in the Liberty ID-WSF Utility schema. The local definition of
503 status codes are described in [Section 4.2](#).

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

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

```
506 <!-- ResponseAbstractType - common message response structure -->
507
508 <xs:complexType name="ResponseAbstractType" abstract="true">
509   <xs:sequence>
510     <xs:element ref="lu:Status"/>
511   </xs:sequence>
512   <xs:anyAttribute namespace="##other" processContents="lax"/>
513 </xs:complexType>
514
```

515 **4.4. Operation: *GetPMEngine***

516 The *GetPMEngine* operation is used by the PMM to obtain the executable code for a PMEEngine that is not currently
517 instantiated within the PMM's domain.

518 **4.4.1. *wsa:Action* values for *GetPMEngine* Messages**

519 <GetPMEngine> request messages MUST include a <wsa:Action> SOAP header with the value of
520 "urn:liberty:prov:2006-12:GetPMEngine".

521 <GetPMEngineResponse> messages MUST include a <wsa:Action> SOAP header with the value of
522 "urn:liberty:prov:2006-12:GetPMEngineResponse".

523 **4.4.2. *GetPMEngine* Message**

524 The <GetPMEngine> request is called to obtain the executable code for a PMEEngine referenced by a
525 <PMEngineRef>.

526 The <prov:GetPMEngine> request contains the following attributes and/or elements:

- 527 • <PMEngineRef> **[Required]** - the reference to the desired PMEEngine. This is typically taken from the element
528 with the same name in a PMD that is being provisioned or updated at the PMM.
- 529 • dp:BasicPagingAttributeGroup **[Optional]** - a set of attributes supporting the pagination of results. This
530 adaptation of the pagination design pattern from the Liberty ID-WSF Design Patterns [\[LibertyDP\]](#) specification
531 uses the Basic Pagination model. The objects being downloaded are already static and so there is no need for the
532 static result set support.
533 For the purpose of pagination, the "item" being paginated is each byte of the executable being downloaded. So a
534 pagination request with a count of 100 is requesting at most 100 bytes of the executable file.
- 535 • anyAttribute **[Optional]** - zero or more attributes from a namespace other than that of this specification. One
536 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

537 The schema for the `<prov:GetPMEngine>` is shown below.

```

538 <!-- GetPMEngine - retrieve the specified PMEEngine -->
539
540 <xs:element name="GetPMEngine" type="GetPMEngineType"/>
541
542 <xs:complexType name="GetPMEngineType">
543   <xs:complexContent>
544     <xs:extension base="RequestAbstractType">
545       <xs:sequence>
546         <xs:element ref="PMEngineRef" />
547       </xs:sequence>
548       <xs:attributeGroup ref="dp:BasicPagingAttributeGroup" />
549     </xs:extension>
550   </xs:complexContent>
551 </xs:complexType>
552
```

553 **Figure 3. `<prov:GetPMEngine>` Schema Fragment**

554 An example message body containing a `<GetPMEngine>` message follows. This request asks for the first 100K bytes
 555 of the executable.

```

556 <prov:GetPMEngine count="102400">
557   <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
558 </prov:GetPMEngine>
559
```

560 **Example 2. Example `<prov:GetPMEngine>` Message**

561 4.4.3. `GetPMEngineResponse` Message

562 This response to the `<prov:GetPMEngine>` request contains the following elements:

- 563 • `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- 564 • `<prov:EngineData>` **[Optional]** - the Base64 encoded bytes of the PMEEngine executable. This MAY contain
 565 only a portion of the complete PMEEngine depending upon the settings for the pagination attributes on the request.
- 566 • `dp:BasicPagingResponseAttributeGroup` **[Optional]** - a set of response attributes supporting the pagina-
 567 tion of results. This adaptation of the pagination design pattern from the Liberty ID-WSF Design Patterns [[Liber-](#)
 568 [tyDP](#)] specification uses the Basic Pagination model.
 569 For the purpose of pagination, the "item" being paginated is each byte of the executable being downloaded. So a
 570 response with the pagination attribute `remaining` set to `8192` indicates that there are 8,192 bytes of data left (after
 571 the bytes included in the current response) to complete the download of the executable file.
- 572 • `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One
 573 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

```

574 <!-- GetPMEngineResponse - response for the GetPMEngine Request -->
575
576 <xs:element name="GetPMEngineResponse" type="GetPMEngineResponseType"/>
577
578 <xs:complexType name="GetPMEngineResponseType">
579   <xs:complexContent>
580     <xs:extension base="RequestAbstractType">
581       <xs:sequence>
582         <xs:element ref="EngineData" minOccurs="0" />
583       </xs:sequence>
584       <xs:attributeGroup ref="dp:BasicPagingResponseAttributeGroup" />
585     </xs:extension>
586   </xs:complexContent>
587 </xs:complexType>
588
589 <xs:element name="EngineData" type="xs:base64Binary" />
590

```

591 **Figure 4. <prov: GetPMEngineResponse> Schema Fragment**

592 An example message body containing a <GetPMEngineResponse> message follows. This is a successful response
593 which includes the first 100K bytes and indicates that there are an additional 83,276 bytes to go in the file.

```

594 <prov: GetPMEngineResponse remaining="83276" offset="102400">
595   <lu: Status code="OK" />
596   <prov: EngineData>
597     ... base64 encoded data (100K worth) ...
598   </prov: EngineData>
599 </prov: GetPMEngineResponse>
600

```

601 **Example 3. Example <prov: GetPMEngineResponse> Message**

602 4.4.4. GetPMEngine Processing Rules

- 603 • All of the processing rules defined for pagination in the Liberty ID-WSF Design Patterns [LibertyDP] are
604 incorporated by reference and must be met.
- 605 • The Provisioning Service SHOULD take steps to ensure that the correct party is submitting the request and reject
606 requests from inappropriate parties.
- 607 • Requests which include a <PMEngineRef> that is not known to the ProvS MUST result in a failure. If detailed
608 status codes are to be included in the response, the detailed error code for this error MUST be "NotFound".
- 609 • If request processing succeeded, the top-level status code MUST be "OK". Otherwise, the top-level status code
610 MUST be "Failed"
- 611 • If the top-level status code is "Failed", the response MAY also contain *Forbidden* as a second-level status code.
612 The Provisioning Service instance may not wish to reveal the reason for failure, in which case no second-level
613 status code will appear.

614 4.5. Operation: *PMArtifactResolve*

615 The *PMArtifactResolve* operation is used by the PMM to exchange a `<PMArtifact>` for a `<PMDescriptor>`.

616 4.5.1. `wsa:Action` values for *PMArtifactResolve* Messages

617 `<PMArtifactResolve>` request messages MUST include a `<wsa:Action>` SOAP header with the value of
 618 "urn:liberty:prov:2006-12:PMArtifactResolve".

619 `<PMArtifactResolveResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of
 620 "urn:liberty:prov:2006-12:PMArtifactResolveResponse".

621 4.5.2. *PMArtifactResolve* Message

622 The `<PMArtifactResolve>` request is called to exchange a `<PMArtifact>` for a `<PMDescriptor>`.

623 The `<prov:PMArtifactResolve>` request contains the following attributes and/or elements:

- 624 • `PMArtifact` **[Required]** - the `PMArtifact` that is being resolved.
- 625 • `<prov:CallbackEPR>` **[Required]** - one or more ID-WSF Endpoint References which describe how the ProvS
 626 can communicate with the PMM for maintenance operations (see [Section 3.5](#)).
- 627 If more than one `<prov:CallbackEPR>` element is present, the ProvS may choose any of the elements present
 628 (recognizing that the list is in preference order – the most preferred element is first).
- 629 • `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
 630 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

631 The schema for the `<prov:PMArtifactResolve>` is shown below.

```

632 <!-- PMArtifactResolve - request to exchange PMHandle for PMDescriptor -->
633
634 <xs:element name="PMArtifactResolve" type="PMArtifactResolveType"/>
635
636 <xs:complexType name="PMArtifactResolveType">
637   <xs:complexContent>
638     <xs:extension base="RequestAbstractType">
639       <xs:sequence>
640         <xs:element ref="PMArtifact" />
641         <xs:element ref="CallbackEPR" />
642       </xs:sequence>
643     </xs:extension>
644   </xs:complexContent>
645 </xs:complexType>
646

```

647 **Figure 5. `<prov:PMArtifactResolve>` — Schema Fragment**

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

```

650 <prov:PMArtifactResolve>
651   <prov:PMArtifact>23asdfhoi323hposdf923h9sdfhweorh2398asdfjweo iha</PMArtifact>
652 </prov:PMArtifactResolve>
653

```

654 **Example 4. Example `<prov:PMArtifactResolve>` Message**

655 4.5.3. PMArtifactResolveResponse Message

656 This response to the <prov:PMArtifactResolve> request contains the following elements:

- 657 • <lu:Status> **[Required]** - the status of the response. See the processing rules below for more information.
- 658 • <prov:PMDescriptor> **[Optional]** - the Provisioned Module Descriptor that is associated with the supplied
- 659 <prov:PMArtifact>. This element will not be present on unsuccessful responses.
- 660 • anyAttribute **[Optional]** - zero or more attributes from a namespace other than that of this specification. One
- 661 such possibility is an **xs:ID** type attribute such as xml:id or wsu:Id.

```

662 <!-- PMArtifactResolveResponse - response to the PMArtifactResolve request -->
663
664 <xs:element name="PMArtifactResolveResponse" type="PMArtifactResolveResponseType"/>
665
666 <xs:complexType name="PMArtifactResolveResponseType">
667   <xs:complexContent>
668     <xs:extension base="ResponseAbstractType">
669       <xs:sequence>
670         <xs:element ref="PMDescriptor" minOccurs="0"/>
671       </xs:sequence>
672     </xs:extension>
673   </xs:complexContent>
674 </xs:complexType>
675

```

676 **Figure 6. <prov:PMArtifactResolveResponse> — Schema Fragment**

677 An example message body containing a <PMArtifactResolveResponse> message follows. This is a successful
678 response.

```

679 <prov:PMArtifactResolveResponse>
680   <lu:Status code="OK" />
681   <prov:PMDescriptor xs:id="2323923900239">
682     <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
683     <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
684     <prov:PMInitData>
685       <xenc:EncryptedData>
686         ... encrypted data here ...
687       </xenc:EncryptedData>
688     </prov:PMInitData>
689     <ds:Signature>
690       ... signature data goes here ...
691     </ds:Signature>
692   </prov:PMDescriptor>
693 </prov:PMArtifactResolveResponse>
694

```

695 **Example 5. Example <prov:PMArtifactResolveResponse> Message**

696 4.5.4. PMArtifactResolve Processing Rules

- 697 • The request **MUST** fail if the PMArtifact is not valid or if it has already been resolved through a previous invocation
- 698 of this interface.
- 699 • The Provisioning Service **SHOULD** take steps to ensure that the correct party is submitting the request and reject
- 700 requests from inappropriate parties.

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

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

706 **4.6. Operation: *PMDActivate***

707 The *PMDActivate* operation is used by a provisioning entity to activate a currently inactive PM.

708 **4.6.1. *wsa:Action* values for *PMDActivate* Messages**

709 <*PMDActivate*> request messages MUST include a <*wsa:Action*> SOAP header with the value of
710 "urn:liberty:prov:2006-12:*PMDActivate*".

711 <*PMDActivateResponse*> messages MUST include a <*wsa:Action*> SOAP header with the value of
712 "urn:liberty:prov:2006-12:*PMDActivateResponse*".

713 **4.6.2. *PMDActivate* Message**

714 The <*PMDActivate*> request is called by a provisioning entity (such as the RegSvc show in the Provisioning
715 workflow in the Liberty ID-WSF Advanced Client Technologies Overview [\[LibertyACT\]](#) to activate an inactive PM.

716 The <*pmm:PMDActivate*> request contains one or more <*pmm:PMDActivateItem*> elements each of which contain
717 the following attributes and/or elements:

718 • <*PMID*> **[Required]** - the identifier of the PM being activated.

719 • *at* **[Optional]** - a time at which the activation should take place. If specified, the value SHOULD be some point
720 in the future at which the PMM would activate the PM.

721 If this attribute is not specified, or it is specified with a time in the past, the PM MUST be activated now.

722 Note that since this request is made to the ProvS and not the PMM, the time it takes to relay the message, especially
723 in a scenario where the PMM is polling the ProvS, may delay the activation time, especially when this attribute
724 has a very short timeframe.

725 The schema for the `<pmm:PMDActivate>` is shown below.

```

726 <!-- PMDActivate - to activate one or more PM(s) at the PMM -->
727
728 <xs:element name="PMDActivate" type="PMDActivateType"/>
729
730 <xs:complexType name="PMDActivateType">
731   <xs:complexContent>
732     <xs:extension base="RequestAbstractType">
733       <xs:sequence>
734         <xs:element ref="PMDActivateItem" maxOccurs="unbounded" />
735       </xs:sequence>
736     </xs:extension>
737   </xs:complexContent>
738 </xs:complexType>
739
740 <xs:element name="PMDActivateItem" type="PMDActivateItemType" />
741
742 <xs:complexType name="PMDActivateItemType">
743   <xs:sequence>
744     <xs:element ref="prov:PMID" />
745   </xs:sequence>
746   <xs:attribute name="at" type="xs:dateTime" use="optional" />
747 </xs:complexType>
748

```

749 **Figure 7. `<pmm:PMDActivate>` — Schema Fragment**

750 An example message body containing a `<pmm:PMDActivate>` message follows. This request activates 2 PMMs, on
 751 at 1PM on the 18th of Dec, 2006 and one now.

```

752 <pmm:PMDActivate>
753   <pmm:PMDActivateItem at="2006-12-18T13:00:00Z" >
754     <prov:PMID issuer="http://provs-r-us.com">uid:239032-230328-92379-2397923</prov:PMID>
755   </pmm:PMDActivateItem>
756   <pmm:PMDActivateItem>
757     <prov:PMID issuer="http://provs-r-us.com">uid:392380-948492-18923-2389238</prov:PMID>
758   </pmm:PMDActivateItem>
759 </pmm:PMDActivate>
760

```

761 **Example 6. Example `<pmm:PMDActivate>` Message**

762 4.6.3. `PMDActivateResponse` Message

763 This response to the `<pmm:PMDActivate>` request contains the following elements:

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

```
767 <!-- PMDActivateResponse - the response to the PMDActivate request -->
768
769 <xs:element name="PMDActivateResponse" type="PMDActivateResponseType"/>
770
771 <xs:complexType name="PMDActivateResponseType">
772   <xs:complexContent>
773     <xs:extension base="ResponseAbstractType" />
774   </xs:complexContent>
775 </xs:complexType>
776
```

777 **Figure 8. <pmm:PMDActivateResponse> — Schema Fragment**

778 An example message body containing a <pmm:PMDActivateResponse> message follows. This is a successful
779 response.

```
780 <pmm:PMDActivate>
781   <lu:Status code="OK" />
782 </pmm:PMDActivate>
783
```

784 **Example 7. Example <pmm:PMDActivateResponse> Message**

785 4.6.4. PMDActivate Processing Rules

- 786 • The ProvS SHOULD return this acknowledgement of receipt to the caller immediately. The caller can then at a
787 later point in time use the <PMDGetStatus> to determine if the activation has completed.
- 788 • Upon receipt of this request, the ProvS MUST change the the status of the PMD to: *urn:liberty:prov:2006-*
789 *12:status:Activating* to indicate that an activation is in progress.
- 790 • If request processing succeeded for all PMs, the top-level status code MUST be *OK*. If the request processing
791 failed for all PMs, the top-level status code MUST be *Failed*. Otherwise, if the results were mixed, the top-level
792 status MUST be *Partial*. and a second-level status MUST be included for the items for which the processing was
793 not successful. The second level status for such items MUST indicate that the processing failed and MUST include
794 the *ref* attribute containing the *PMID* value for the item. These second-level status codes MAY simply be *Failed*,
795 or they may indicate with more detail the reason for the failure.
- 796 • If the top-level status is not *OK*. and second level status codes are present, they MAY contain detailed error
797 information if the PMM wants to share that information with the invoking party.

798 **4.7. Operation: *PMDDeactivate***

799 The *PMDDeactivate* operation is used by a provisioning entity to deactivate a currently active PM.

800 **4.7.1. *wsa:Action* values for *PMDDeactivate* Messages**

801 <PMDDeactivate> request messages MUST include a <*wsa:Action*> SOAP header with the value of
802 "urn:liberty:prov:2006-12:PMDDeactivate".

803 <PMDDeactivateResponse> messages MUST include a <*wsa:Action*> SOAP header with the value of
804 "urn:liberty:prov:2006-12:PMDDeactivateResponse".

805 **4.7.2. *PMDDeactivate* Message**

806 The <PMDDeactivate> request is called by a provisioning entity (such as the RegSvc show in the Provisioning
807 workflow in the Liberty ID-WSF Advanced Client Technologies Overview [[LibertyACT](#)]) to deactivate an activated
808 PM.

809 The <*pmm:PMDDeactivate*> request contains one or more <*pmm:PMDDeactivateItem*> elements each of which
810 contain the following attributes and/or elements:

- 811 • <PMID> [**Required**] - the identifier of the PM being deactivated.
- 812 • at [**Optional**] - a time at which the deactivation should take place. If specified, the value SHOULD be some point
813 in the future at which the PMM would deactivate the PM.
- 814 If this attribute is not specified, or it is specified with a time in the past, the PM MUST be deactivated now.
- 815 Note that since this request is made to the ProvS and not the PMM, the time it takes to relay the message, especially
816 in a scenario where the PMM is polling the ProvS, may delay the deactivation time, especially when this attribute
817 has a very short timeframe.

818 The schema for the <*pmm:PMDDeactivate*> is shown below.

```
819 <!-- PMDDeactivate - to deactivate a PM at the PMM -->
820
821 <xs:element name="PMDDeactivate" type="PMDDeactivateType"/>
822
823 <xs:complexType name="PMDDeactivateType">
824   <xs:complexContent>
825     <xs:extension base="RequestAbstractType">
826       <xs:sequence>
827         <xs:element ref="PMDDeactivateItem" maxOccurs="unbounded" />
828       </xs:sequence>
829     </xs:extension>
830   </xs:complexContent>
831 </xs:complexType>
832
833 <xs:element name="PMDDeactivateItem" type="PMDDeactivateItemType" />
834
835 <xs:complexType name="PMDDeactivateItemType">
836   <xs:sequence>
837     <xs:element ref="prov:PMID" />
838   </xs:sequence>
839   <xs:attribute name="at" type="xs:dateTime" use="optional" />
840 </xs:complexType>
841
```

842 **Figure 9. <*pmm:PMDDeactivate*> — Schema Fragment**

843 An example message body containing a `<pmm:PMDDeactivate>` message follows. This request deactivates two
 844 PMs, one now and one near midnight New Year's eve.

```
845 <pmm:PMDDeactivate>
846   <pmm:PMDDeactivateItem at="2006-12-31T23:59:59Z" >
847     <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
848   </pmm:PMDDeactivateItem>
849   <pmm:PMDDeactivateItem>
850     <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-18923-2389238</prov:PMID>
851   </pmm:PMDDeactivateItem>
852 </pmm:PMDDeactivate>
853
```

854 **Example 8. Example `<pmm:PMDDeactivate>` Message**

855 4.7.3. `PMDDeactivateResponse` Message

856 This response to the `<pmm:PMDDeactivate>` request contains the following elements:

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

```
860 <!-- PMDDeactivateResponse - the response to the PMDDeactivate request -->
861
862 <xs:element name="PMDDeactivateResponse" type="PMDDeactivateResponseType"/>
863
864 <xs:complexType name="PMDDeactivateResponseType">
865   <xs:complexContent>
866     <xs:extension base="ResponseAbstractType" />
867   </xs:complexContent>
868 </xs:complexType>
869
```

870 **Figure 10. `<pmm:PMDDeactivateResponse>` — Schema Fragment**

871 An example message body containing a `<pmm:PMDDeactivateResponse>` message follows. This is a partially
 872 successful response where one of the PMs were not found.

```
873 <pmm:PMDDeactivateResponse>
874   <lu:Status code="Partial">
875     <lu:Status ref="uuid:239032-230328-92379-2397923" code="NotFound" />
876   </lu:Status>
877 </pmm:PMDDeactivateResponse>
878
```

879 **Example 9. Example `<pmm:PMDDeactivateResponse>` Message**

880 4.7.4. `PMDDeactivate` Processing Rules

- 881 • The ProvS SHOULD return this acknowledgement of receipt to the caller immediately. The caller can then at a
 882 later point in time use the `<PMDGetStatus>` to determine if the deactivation has completed.
- 883 • Upon receipt of this request, the ProvS MUST change the the status of the PMD to: `urn:liberty:prov:2006-`
 884 `12:status:Deactivating` to indicate that a deactivation is in progress.

- 885 • If request processing succeeded for all PMs, the top-level status code MUST be *OK*. If the request processing
886 failed for all PMs, the top-level status code MUST be *Failed*. Otherwise, if the results were mixed, the top-level
887 status MUST be *Partial*. and a second-level status MUST be included for the items for which the processing was
888 not successful. The second level status for such items MUST indicate that the processing failed and MUST include
889 the `ref` attribute containing the `PMID` value for the item. These second-level status codes MAY simply be *Failed*,
890 or they may indicate with more detail the reason for the failure.
- 891 • If the top-level status is not *OK*. and second level status codes are present, they MAY contain detailed error
892 information if the PMM wants to share that information with the invoking party.

893 4.8. Operation: *PMDelete*

894 The *PMDelete* operation is used by a PM issuing party (such as an IdP) to cancel previously registered PMDs at the
895 ProvS. The previously registered PMDs MAY have already been provisioned into running PMs (in which case, the
896 ProvS would initiate an deletion process with the PMM where the PM has been provisioned).

897 4.8.1. `wsa:Action` values for *PMDelete* Messages

898 `<PMDelete>` request messages MUST include a `<wsa:Action>` SOAP header with the value of
899 `"urn:liberty:prov:2006-12:PMDelete"`.

900 `<PMDeleteResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of
901 `"urn:liberty:prov:2006-12:PMDeleteResponse"`.

902 4.8.2. *PMDelete* Message

903 The `<PMDelete>` request is called to cancel a PMD that was registered with the ProvS.

904 The `<prov:PMDelete>` request contains the following elements/attributes:

- 905 • `<PMID>` [**Required**] - one or more identifiers for the `PMDescriptors` that are to be deleted. This element MUST
906 contain a value that matches the `PMID` of a previously registered `<PMDescriptor>`.
- 907 • `anyAttribute` [**Optional**] - zero or more attributes from a namespace other than that of this specification. One
908 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

909 The schema for the `<prov:PMDelete>` is shown below.

```
910 <!-- PMDelete - to register a new PMD at the ProvS -->
911
912 <xs:element name="PMDelete" type="PMDeleteType"/>
913 <xs:complexType name="PMDeleteType">
914   <xs:complexContent>
915     <xs:extension base="RequestAbstractType">
916       <xs:sequence>
917         <xs:element ref="PMID" maxOccurs="unbounded" />
918       </xs:sequence>
919     </xs:extension>
920   </xs:complexContent>
921 </xs:complexType>
922
```

923 **Figure 11.** `<prov:PMDelete>` — Schema Fragment

924 An example message body containing a `<prov:PMDelete>` message follows.

```

925 <prov:PMDelete>
926   <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
927   <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-18923-2389238</prov:PMID>
928 </prov:PMDelete>
929
  
```

930 **Example 10. Example <prov:PMDelete> Message**

931 4.8.3. PMDeleteResponse Message

932 This response to the <prov:PMDelete> request contains the following elements/attributes:

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

```

936 <!-- PMDeleteResponse - the response to the PMDelete request -->
937
938 <xs:element name="PMDeleteResponse" type="PMDeleteResponseType"/>
939
940 <xs:complexType name="PMDeleteResponseType">
941   <xs:complexContent>
942     <xs:extension base="ResponseAbstractType" />
943   </xs:complexContent>
944 </xs:complexType>
945
  
```

946 **Figure 12. <prov:PMDeleteResponse> — Schema Fragment**

947 An example message body containing a <PMDeleteResponse> message follows. In this case, the response

948 indicates a partial success and the secondary codes indicate that the second PMID was not found.

```

949 <prov:PMDCancelResponse>
950   <lu:Status code="Partial">
951     <lu:Status ref="uuid:392380-948492-18923-2389238" code="NotFound" />
952   </lu:Status>
953 </prov:PMDCancelResponse>
954
  
```

955 **Example 11. Example <prov:PMDeleteResponse> Message**

956 4.8.4. PMDelete Processing Rules

- 957 • The invoking party **MUST** specify a value for the <PMID> element in the <PMDescriptor> that matches a value
- 958 previously used **by this issuing party**. If this is not the case, the ProvS **MUST** reject the delete request and, if the
- 959 ProvS includes secondary status codes in the response, **MUST** set the secondary status code to *NotFound*.
- 960 • If all <prov:PMDeleteItem> requests are successfully processed, the top-level status code **MUST** be *"OK"*.
- 961 If all of the items failed, the the top-level status code **MUST** be *"Failed"*. Otherwise, if the results were mixed,
- 962 the top-level status **MUST** be *"Partial"* and the second level status **MUST** be included for items for which the
- 963 processing was not successful indicating so and including the `ref` attribute containing the <PMID> value for the
- 964 item. These second-level status codes **MAY** simply be *"Failed"*, or they may indicate with more detail the reason
- 965 for the failure.

- 966 • If the top-level status code is "Failed", the response MAY also contain other status codes (such as *NotFound* or
967 *FeatureNotSupported*) as a second-level status code. The ProvS instance may not wish to reveal the reason for
968 failure, in which case no second-level status code will appear.

969 4.9. Operation: *PMDGetStatus*

970 The *PMDGetStatus* operation is used by a PM issuing party (such as an IdP) to obtain the current status of an issued
971 PM.

972 4.9.1. *wsa:Action* values for *PMDGetStatus* Messages

973 <*PMDGetStatus*> request messages MUST include a <*wsa:Action*> SOAP header with the value of
974 "urn:liberty:prov:2006-12:*PMDGetStatus*".

975 <*PMDGetStatusResponse*> messages MUST include a <*wsa:Action*> SOAP header with the value of
976 "urn:liberty:prov:2006-12:*PMDGetStatusResponse*".

977 4.9.2. *PMDGetStatus* Message

978 The <*PMDGetStatus*> request is called to obtain the status of a PMD that was registered with the ProvS.

979 The <*prov:PMDGetStatus*> request contains the following elements/attributes:

- 980 • <PMID> [**Required**] - one or more identifiers for the PMs who's status is requested. This element MUST contain
981 a value that matches the PMID of a previously registered <*PMDDescriptor*>.
- 982 • *anyAttribute* [**Optional**] - zero or more attributes from a namespace other than that of this specification. One
983 such possibility is an *xs:ID* type attribute such as *xml:id* or *wsu:Id*.

984 The schema for the <*prov:PMDGetStatus*> is shown below.

```
985 <!-- PMDStatus - to get the status of a PMD at the ProvS -->
986
987 <xs:element name="PMDGetStatus" type="PMDGetStatusType"/>
988
989 <xs:complexType name="PMDGetStatusType">
990   <xs:complexContent>
991     <xs:extension base="RequestAbstractType">
992       <xs:sequence>
993         <xs:element ref="PMID" maxOccurs="unbounded" />
994       </xs:sequence>
995     </xs:extension>
996   </xs:complexContent>
997 </xs:complexType>
998
```

999 **Figure 13.** <*prov:PMDGetStatus*> — Schema Fragment

1000 An example message body containing a <*prov:PMDGetStatus*> message follows.

```
1001 <prov:PMDGetStatus>
1002   <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
1003   <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-18923-2389238</prov:PMID>
1004 </prov:PMDGetStatus>
1005
```

1006 **Example 12.** Example <*prov:PMDGetStatus*> Message

1007 **4.9.3. PMDGetStatusResponse Message**

1008 This response to the <prov:PMDGetStatus> request contains the following elements/attributes:

- 1009 • <lu:Status> **[Required]** - the status of the response. See the processing rules below for more information.
- 1010 • <prov:PMStatus> **[Required]** - zero or more elements describing the status of the requested PMs.
- 1011 • anyAttribute **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
- 1012 such possibility is an **xs:ID** type attribute such as xml:id or wsu:Id.

```

1013 <!-- PMDStatusResponse - response to the PMDStatus request -->
1014
1015 <xs:element name="PMDStatusResponse" type="PMDStatusResponseType"/>
1016
1017 <xs:complexType name="PMDStatusResponseType">
1018   <xs:complexContent>
1019     <xs:extension base="ResponseAbstractType">
1020       <xs:sequence>
1021         <xs:element ref="PMStatus" minOccurs="0"/>
1022       </xs:sequence>
1023     </xs:extension>
1024   </xs:complexContent>
1025 </xs:complexType>
1026

```

1027 **Figure 14. <prov:PMDGetStatusResponse> — Schema Fragment**

1028 An example message body containing a <PMDGetStatusResponse> message follows. In this successful response,
 1029 the status of one PM is running and the other PM was not found.

```

1030 <prov:PMDGetStatusResponse>
1031   <lu:Status code="OK" />
1032   <prov:PMStatus>
1033     <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
1034     <prov:State asof="2006-12-14T17:31:11Z">urn:liberty:prov:2006-12:status:Activated</prov:
1035 State>
1036   </prov:PMStatus>
1037   <prov:PMStatus>
1038     <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-18923-2389238</prov:PMID>
1039     <prov:State>urn:liberty:prov:2006-12:status:NotFound</prov:State>
1040   </prov:PMStatus>
1041 </prov:PMDGetStatusResponse>
1042

```

1043 **Example 13. Example <prov:PMDGetStatusResponse> Message**

1044 **4.9.4. PMDGetStatus Processing Rules**

- 1045 • The invoking party **MUST** specify a value for the <PMID> elements that match the value of existing, registered
 1046 <PMDescriptor>s. If this is not the case, the ProvS **MUST** treat this status request as a failure. If detailed
 1047 second-level status codes are include in the response, the status code for this entry **MUST** be set to *NotFound*.
- 1048 • If all operations are successfully processed, the top-level status code **MUST** be "OK". If all of the operations failed,
 1049 the the top-level status code **MUST** be "Failed". Otherwise, if the results were mixed, the top-level status **MUST**
 1050 be "Partial" and the second-level status entries **MUST** be included for the operations for which the processing
 1051 was not successful indicating so and including the ref attribute containing the <PMID> value for the unsuccessful
 1052 operation.

- 1053 • If the top-level status code is "Failed", or "Partial", the response MAY also contain more detailed second-level
1054 status codes (such as *NotFound*). The ProvS instance may not wish to reveal the reason for failure, in which case
1055 second-level status code might not be present or, for the case of a "Partial" top-level status, the second-level status
1056 MAY simply be "Failed".

1057 4.10. Operation: *PMDRegister*

1058 The *PMDRegister* operation is used by a PM issuing party (such as an IdP) to register PMDs with the ProvS and obtain
1059 the PHs that it can issue to the PMM.

1060 4.10.1. wsa:Action values for *PMDRegister* Messages

1061 <PMDRegister> request messages MUST include a <wsa:Action> SOAP header with the value of
1062 "urn:liberty:prov:2006-12:PMDRegister".

1063 <PMDRegisterResponse> messages MUST include a <wsa:Action> SOAP header with the value of
1064 "urn:liberty:prov:2006-12:PMDRegisterResponse".

1065 4.10.2. *PMDRegister* Message

1066 The <PMDRegister> request is called to register a new PMD with the ProvS.

1067 The <prov:PMDRegister> request contains one or more <prov:PMDRegisterItem> elements which have
1068 the following contents:

- 1069 • <PMDDescriptor> [**Required**] - the PMD that is being registered.
- 1070 • itemID [**Required**] - the identifier for this request item (for correlation within the results).

1071 The schema for the <prov:PMDRegister> is shown below.

```
1072 <!-- PMDRegister - to register a new PMD at the ProvS -->
1073
1074 <xs:element name="PMDRegister" type="PMDRegisterType"/>
1075
1076 <xs:complexType name="PMDRegisterType">
1077   <xs:complexContent>
1078     <xs:extension base="RequestAbstractType">
1079       <xs:sequence>
1080         <xs:element ref="PMDRegisterItem" maxOccurs="unbounded" />
1081       </xs:sequence>
1082     </xs:extension>
1083   </xs:complexContent>
1084 </xs:complexType>
1085
1086 <xs:element name="PMDRegisterItem" type="PMDRegisterItemType" />
1087
1088 <xs:complexType name="PMDRegisterItemType">
1089   <xs:sequence>
1090     <xs:element ref="PMDDescriptor" />
1091   </xs:sequence>
1092   <xs:attribute name="itemID" type="xs:string" use="required" />
1093 </xs:complexType>
1094
```

1095 **Figure 15.** <prov:PMDRegister> — Schema Fragment

1096 An example message body containing a <prov:PMDRegister> message follows.

```

1097 <prov:PMDRegister>
1098   <prov:PMDRegisterItem itemID="1">
1099     <prov:PMDDescriptor xs:id="2323923900239">
1100       <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
1101       <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
1102       <prov:PMInitData>
1103         <xenc:EncryptedData>
1104           .... encrypted data here ...
1105         </xenc:EncryptedData>
1106       </prov:PMInitData>
1107       <ds:Signature>
1108         ... signature data goes here ...
1109       </ds:Signature>
1110     </prov:PMDDescriptor>
1111   </prov:PMDRegisterItem>
1112 </prov:PMDRegister>
1113

```

1114 **Example 14. Example <prov:PMDRegister> Message**

1115 4.10.3. PMDRegisterResponse Message

1116 This response to the <prov:PMDRegister> request contains the following elements/attributes:

- 1117 • <lu:Status> **[Required]** - the status of the response. See the processing rules below for more information.
- 1118 • <prov:PMDRegisterResponseItem> **[Optional]** - zero or more response items which are included for for
 1119 successful responses. If present, this element contains the following content:
 - 1120 • <prov:PMID> **[Required]** - the Provisioned Module IDentifier assigned to this newly registered PMD.
 - 1121 • <prov:ProvisioningHandle> **[Required]** - the Provisioning Handle created by the ProvS to provide a
 1122 PMM with the information necessary to obtain the PMD that was registered.
 - 1123 • ref **[Required]** - a reference to the itemID of the <prov:PMDRegisterItem> element that this
 1124 <prov:PMDRegisterResponseItem> is in response to.
 - 1125 • anyAttribute **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
 1126 such possibility is an **xs:ID** type attribute such as xml:id or wsu:Id.

```

1127 <!-- PMDRegisterResponse - response to the PMDRegister request -->
1128
1129 <xs:element name="PMDRegisterResponse" type="PMDRegisterResponseType"/>
1130
1131 <xs:complexType name="PMDRegisterResponseType">
1132   <xs:complexContent>
1133     <xs:extension base="ResponseAbstractType">
1134       <xs:sequence>
1135         <xs:element ref="PMDRegisterResponseItem" minOccurs="0"/>
1136       </xs:sequence>
1137     </xs:extension>
1138   </xs:complexContent>
1139 </xs:complexType>
1140
1141 <xs:element name="PMDRegisterResponseItem"
1142   type="PMDRegisterResponseItemType"/>
1143
1144 <xs:complexType name="PMDRegisterResponseItemType">
1145   <xs:sequence>
1146     <xs:element ref="ProvisioningHandle" />
1147   </xs:sequence>
1148   <xs:attribute name="ref" type="xs:string" use="required" />
1149 </xs:complexType>
1150

```

1151 **Figure 16. <prov:PMDRegisterResponse> — Schema Fragment**

1152 An example message body containing a <PMDRegisterResponse> message follows.

```

1153 <prov:PMDRegisterResponse>
1154   <lu:Status code="OK" />
1155   <prov:PMDRegisterResponseItem ref="1">
1156     <prov:ProvisioningHandle xs:id="2302384823023">
1157       <prov:PMArtifact>23asdfhoi323hposdf923h9sdfhweorh2398asdfjweoiha</prov:PMArtifact >
1158       <prov:ProvisioningServiceEPR>
1159         <wsa:Address>http://provision.idpsRus.com</wsa:Address>
1160         <wsa:Metadata>
1161           <ds:Abstract>Provisioning Service</ds:Abstract>
1162           <ds:ProviderID>http://provisioning-provider.idpsRus.com/</ds:ProviderID>
1163           <ds:ServiceType>urn:liberty:prov:2006-12</ds:ServiceType>
1164           <ds:Framework version="2.0" />
1165           <ds:SecurityContext>
1166             <ds:SecurityMechID>
1167               urn:liberty:security:2005-02:TLS:SAMLV2
1168             </ds:SecurityMechID>
1169             <sec:Token ref="urn:liberty:disco:tokenref:ObtainFromIDP" />
1170           </ds:SecurityContext>
1171         </wsa:Metadata>
1172       </prov:ProvisioningServiceEPR>
1173       <ds:Signature>
1174         ... signature info here ..
1175       </ds:Signature>
1176     </prov:ProvisioningHandle>
1177   </prov:PMDRegisterResponseItem>
1178 </prov:PMDRegisterResponse>
1179

```

1180 **Example 15. Example <prov:PMDRegisterResponse> Message**

1181 **4.10.4. PMDRegister Processing Rules**

- 1182 • The invoking party **MUST NOT** specify a value for the <PMID> element in the <PMDescriptor> that matches
 1183 a value previously used. If this does happen, the ProvS **MUST** reject the registration request and, if the ProvS
 1184 includes secondary status codes in the response, **MUST** set the status code to *AlreadyInUse*.

- 1185 • Each <prov:PMDRegisterItem> is processed independently and may independently succeed or fail on its
 1186 own merits.

- 1187 • If all <prov:PMDRegisterItem> requests are successfully processed, the top-level status code **MUST** be
 1188 "OK". If all of the items failed, the the top-level status code **MUST** be "Failed". Otherwise, if the results were
 1189 mixed, the top-level status **MUST** be "Partial" and the second level status **MUST** be included for items for which
 1190 the processing was not successful indicating so and including the ref attribute containing the itemID value for
 1191 the item. These second-level status codes **MAY** simply be "Failed", or they may indicate with more detail the
 1192 reason for the failure.

- 1193 • If the top-level status code is "Failed", the response **MAY** also contain other status codes (such as *NotFound* or
 1194 *FeatureNotSupported*) as a second-level status code. The ProvS instance may not wish to reveal the reason for
 1195 failure, in which case no second-level status code will appear.

1196 4.11. Operation: *PMDSetStatus*

1197 The *PMDSetStatus* operation is used by the PMM to notify the Provisioning service of the status of the PM (whether
1198 it was successfully provisioned or not).

1199 4.11.1. *wsa:Action* values for *PMDSetStatus* Messages

1200 <*PMDSetStatus*> request messages MUST include a <*wsa:Action*> SOAP header with the value of
1201 "urn:liberty:prov:2006-12:*PMDSetStatus*".

1202 <*PMDSetStatusResponse*> messages MUST include a <*wsa:Action*> SOAP header with the value of
1203 "urn:liberty:prov:2006-12:*PMDSetStatusResponse*".

1204 4.11.2. *PMDSetStatus* Message

1205 The <*PMDSetStatus*> request is called to notify the Provisioning Service about the status of the provisioning of a
1206 PM.

1207 The <*prov:PMDSetStatus*> request contains following attributes and/or elements:

1208 • <*PMStatus*> - one or more status elements (see [Section 3.4](#)) containing the <*PMID*> of the PM and the new
1209 <*State*>.

1210 • *anyAttribute* [Optional] Zero or more attributes from a namespace other than that of this specification. One
1211 such possibility is an *xs:ID* type attribute such as *xml:id* or *wsu:Id*.

1212 The schema for the <*prov:PMDSetStatus*> is shown below.

```
1213 <!-- PMDSetStatus - update provisioning status of PM -->
1214
1215 <xs:element name="PMDSetStatus" type="PMDSetStatusType"/>
1216
1217 <xs:complexType name="PMDSetStatusType">
1218   <xs:complexContent>
1219     <xs:extension base="RequestAbstractType">
1220       <xs:sequence>
1221         <xs:element ref="PMStatus" />
1222       </xs:sequence>
1223     </xs:extension>
1224   </xs:complexContent>
1225 </xs:complexType>
1226
```

1227 **Figure 17. <*prov:PMDSetStatus*> — Schema Fragment**

1228 An example message body containing a <*PMDSetStatus*> message follows. This request shows that the PM has
1229 been installed.

```
1230 <prov:PMDSetStatus>
1231   <prov:PMStatus>
1232     <prov:PMID issuer="http://provs-r-us.com">uu id:239032-230328-92379-239792 3</prov:PMID>
1233     <prov:State asof="2006-12-14T17:31:11Z">urn:liberty:prov:2006-12:status:Activated</prov:Stat
1234 e>
1235   </prov:PMStatus>
1236 </prov:PMDSetStatus>
1237
```

1238 **Example 16. Example <*prov:PMDSetStatus*> Message**

1239 4.11.3. PMDSetStatusResponse Message

1240 This response to the `<prov:PMDSetStatus>` request contains the following elements:

- 1241 • `<lu:Status>` [**Required**] - the status of the response. See the processing rules below for more information.
- 1242 • `anyAttribute` [**Optional**] - zero or more attributes from a namespace other than that of this specification. One
- 1243 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

```
1244 <!-- PMDSetStatusResonse - response for PMDSetStatus request -->
1245
1246 <xs:element name="PMDSetStatusResponse" type="PMDSetStatusResponseType"/>
1247
1248 <xs:complexType name="PMDSetStatusResponseType">
1249   <xs:complexContent>
1250     <xs:extension base="ResponseAbstractType" />
1251   </xs:complexContent>
1252 </xs:complexType>
1253
```

1254 **Figure 18. `<prov:PMDSetStatusResponse>` — Schema Fragment**

1255 An example message body containing a `<PMDSetStatusResponse>` message follows. This is a successful
1256 response.

```
1257 <prov:PMDSetStatusResponse>
1258   <lu:Status code="OK" />
1259 </prov:PMDSetStatusResponse>
1260
```

1261 **Example 17. Example `<prov:PMDSetStatusResponse>` Message**

1262 4.11.4. PMDSetStatus Processing Rules

- 1263 • If all status updates are successfully processed, the top-level status code **MUST** be `"OK"`. If all of the updates failed,
1264 the the top-level status code **MUST** be `"Failed"`. Otherwise, if the results were mixed, the top-level status **MUST**
1265 be `"Partial"` and the second-level status entries **MUST** be included for status updates for which the processing was
1266 not successful indicating so and including the `ref` attribute containing the `PMID` value for the status update.
- 1267 • If the top-level status code is `"Failed"`, or `"Partial"`, the response **MAY** also contain more detailed second-level
1268 status codes (such as `NotFound` or `Forbidden`). The ProvS instance may not wish to reveal the reason for failure,
1269 in which case second-level status code might not be present or, for the case of a `"Partial"` top-level status, the
1270 second-level status **MAY** simply be `"Failed"`.

1271 **4.12. Operation: *PMDUpdate***

1272 The *PMDUpdate* operation is used by a PM issuing party (such as an IdP) to update previously registered PMDs at
1273 the ProvS. The previously registered PMDs MAY have already been provisioned into running PMs (in which case,
1274 the ProvS would initiate an update process with the PMM where the PM has been provisioned).

1275 **4.12.1. *wsa:Action* values for *PMDUpdate* Messages**

1276 <*PMDUpdate*> request messages MUST include a <*wsa:Action*> SOAP header with the value of
1277 "urn:liberty:prov:2006-12:*PMDUpdate*".

1278 <*PMDUpdateResponse*> messages MUST include a <*wsa:Action*> SOAP header with the value of
1279 "urn:liberty:prov:2006-12:*PMDUpdateResponse*".

1280 **4.12.2. *PMDUpdate* Message**

1281 The <*PMDUpdate*> request is called to update a PMD that was registered with the ProvS.

1282 The <prov:*PMDUpdate*> request contains one or more <prov:*PMDUpdateItem*> elements each of which have
1283 the following contents:

1284 • <*PMDDescriptor*> [**Required**] - the updated PMD information. The <*PMID*> element MUST contain a value that
1285 matches the <*PMID*> of a previously registered <*PMDDescriptor*>.

1286 In some cases only a portion of the PM is being updated (such as a *PMEngine* update). Even so, the entire
1287 replacement <*PMDDescriptor*> must be specified (as opposed to just the changed elements).

1288 • *type* [**Required**] - the type of update being applied. The following values are defined for this attribute:

1289 • *urn:liberty:prov:2006-12:ut:replace* - a complete replacement of the existing PMD (and if it has been
1290 provisioned, the PM itself).

1291 • *urn:liberty:prov:2006-12:ut:engine* - an update of the engine only.

1292 • *urn:liberty:prov:2006-12:ut:initdata* - an update of the initialization data

1293 • *itemID* [**Required**] - the identifier for this request item (for correlation within the results).

1294 The schema for the `<prov:PMDUpdate>` is shown below.

```

1295 <!-- PMDUpdate - update the PMD for a existing PM at the ProvS -->
1296
1297 <xs:element name="PMDUpdate" type="PMDUpdateType"/>
1298
1299 <xs:complexType name="PMDUpdateType">
1300   <xs:complexContent>
1301     <xs:extension base="RequestAbstractType">
1302       <xs:sequence>
1303         <xs:element ref="PMDUpdateItem" maxOccurs="unbounded" />
1304       </xs:sequence>
1305     </xs:extension>
1306   </xs:complexContent>
1307 </xs:complexType>
1308
1309 <xs:element name="PMDUpdateItem" type="PMDUpdateItemType" />
1310
1311 <xs:complexType name="PMDUpdateItemType">
1312   <xs:sequence>
1313     <xs:element ref="PMDDescriptor" />
1314   </xs:sequence>
1315   <xs:attribute name="type" type="xs:anyURI" use="required" />
1316   <xs:attribute name="itemID" type="xs:string" use="required" />
1317 </xs:complexType>
1318

```

1319 **Figure 19. `<prov:PMDUpdate>` — Schema Fragment**

1320 An example message body containing a `<prov:PMDUpdate>` message follows. This is an update of the PM Engine
 1321 to version 4.0.

```

1322 <prov:PMDUpdate>
1323   <prov:PMDUpdateItem itemID="1" type="urn:liberty:prov:2006-12:ut:engine">
1324     <prov:PMDDescriptor xs:id="2323923900239" >
1325       <prov:PMID issuer="http://provs-r-us.com">uuid:778349-283920-88379-5448739</prov:PMID>
1326       <prov:PMEngineRef>https://pmsRus.org/VeryTrust edModule/4.0</prov:PMEngineRe f>
1327       <ds:Signature>
1328         ... signature data goes here ...
1329       </ds:Signature>
1330     </prov:PMDDescriptor>
1331   </prov:PMDUpdateItem>
1332 </prov:PMDUpdate>
1333

```

1334 **Example 18. Example `<prov:PMDUpdate>` Message**

1335 4.12.3. `PMDUpdateResponse` Message

1336 This response to the `<prov:PMDUpdate>` request contains the following elements/attributes:

- 1337 • `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- 1338 • `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
 1339 such possibility is an `xs:ID` type attribute such as `xml:id` or `wsu:Id`.

```

1340 <!-- PMDUpdateResponse - response to the PMDUpdate request -->
1341
1342 <xs:element name="PMDUpdateResponse" type="PMDUpdateResponseType"/>
1343
1344 <xs:complexType name="PMDUpdateResponseType">
1345   <xs:complexContent>
1346     <xs:extension base="ResponseAbstractType">
1347       <xs:sequence>
1348         </xs:sequence>
1349     </xs:extension>
1350   </xs:complexContent>
1351 </xs:complexType>
1352

```

1353 **Figure 20.** `<prov:PMDUpdateResponse>` — Schema Fragment

1354 An example message body containing a `<PMDUpdateResponse>` message follows. In this case, the response
 1355 indicates a failure and the secondary code indicates the PMID was not found.

```

1356 <prov:PMDUpdateResponse>
1357   <lu:Status code="Failed">
1358     <lu:Status ref="1" code="NotFound" />
1359   </lu:Status>
1360 </prov:PMDUpdateResponse>
1361

```

1362 **Example 19.** Example `<prov:PMDUpdateResponse>` Message

1363 4.12.4. PMDUpdate Processing Rules

- 1364 • The invoking party **MUST** specify a value for the `<PMID>` element in the `<PMDDescriptor>` that matches a value
 1365 previously used **by this issuing party**. If this is not the case, the ProvS **MUST** reject the update request and, if the
 1366 ProvS does include secondary status codes in the response, **MUST** set the secondary status code to *NotFound*.
- 1367 • Each `<prov:PMDUpdateItem>` is processed independently and may independently succeed or fail on its own
 1368 merits.
- 1369 • If all `<prov:PMDUpdateItem>` requests are successfully processed, the top-level status code **MUST** be *OK*.
 1370 If all of the items failed, the the top-level status code **MUST** be *Failed*. Otherwise, if the results were mixed,
 1371 the top-level status **MUST** be *Partial* and the second level status **MUST** be included for items for which the
 1372 processing was not successful indicating so and including the `ref` attribute containing the `itemID` value for the
 1373 item. These second-level status codes **MAY** simply be *Failed*, or they may indicate with more detail the reason
 1374 for the failure.
- 1375 • If the top-level status code is *Failed*, the response **MAY** also contain other status codes (such as *NotFound* or
 1376 *FeatureNotSupported*) as a second-level status code. The ProvS instance may not wish to reveal the reason for
 1377 failure, in which case no second-level status code will appear.

1378 **4.13. Operation: *Poll***

1379 The *Poll* operation is used by the PMM to poll for any new provisioning maintenance requests when the PMM is
 1380 unable to expose an externally visible endpoint for direct access by the ProvS.

1381 This operation is an adaptation of the *Poll* design pattern and inherits all of its structure and processing rules.

1382 **4.13.1. *wsa:Action* values for *Poll* Messages**

1383 *<Poll>* messages MUST include a *<wsa:Action>* SOAP header with the value of "urn:liberty:prov:2006-12:Poll".

1384 *<PollResponse>* messages MUST include a *<wsa:Action>* SOAP header with the value of
 1385 "urn:liberty:prov:2006-12:PollResponse".

1386 **4.13.2. *Poll* Message**

1387 The *<Poll>* request is called by the PMM to ask the ProvS for any queued provisioning maintenance requests and
 1388 to return any responses to prior requests.

1389 The structure of the *<prov:Poll>* request is derived from the structure of the *<dp:PollType>* without
 1390 modification. See [LibertyDP] for a complete description of the structure and meaning of the elements.

1391 The schema for the *<prov:Poll>* is shown below.

```
1392 <!-- Poll - Poll for new service requests -->
1393
1394 <xs:element name="Poll" type="dp:PollType"/>
1395
```

1396 **Figure 21. *<prov:Poll>* — Schema Fragment**

1397 An example message body containing a *<prov:Poll>* message follows. This is a request asking for
 1398 *<pmm:UpdatePM>* or *<pmm>DeletePM>* requests and asks ProvS to wait for 5 minutes if none are immedi-
 1399 ately available.

```
1400 <prov:Poll wait="300">
1401 <wsa:Action>urn:liberty:pmm:2006-12:PMUpdate</wsa:Action>
1402 <wsa:Action>urn:liberty:pmm:2006-12:PMDelete</wsa:Action>
1403 <wsa:Action>urn:liberty:pmm:2006-12:PMGetStatus</wsa:Action>
1404 </prov:Poll>
1405
```

1406 **Example 20. Example *<prov:Poll>* Message**

1407 Another example message body containing a *<prov:Poll>* message follows. This example also includes a
 1408 *<prov:UpdatePMResponse>* from a prior request received at the PMM.

```
1409 <prov:Poll>
1410 <wsa:Action>urn:liberty:pmm:2006-12:PMUpdate</wsa:Action>
1411 <wsa:Action>urn:liberty:pmm:2006-12:PMDelete</wsa:Action>
1412 <wsa:Action>urn:liberty:pmm:2006-12:PMGetStatus</wsa:Action>
1413 <dp:Response ref="1">
1414 <pmm:UpdatePMResponse>
1415 <lu:Status code="OK" />
1416 </pmm:UpdatePMResponse>
1417 </dp:Response>
1418 </prov:Poll>
1419
```

1420 **Example 21. Example <prov:Poll> Message with a <prov:UpdatePMResponse>.**

1421 4.13.3. PollResponse Message

1422 This response to the <prov:Poll> request is derived from the <dp:PollResponseType> without modification.
 1423 See [LibertyDP] for a complete description of the structure and meaning of the elements.

```
1424 <!--PollResponse - response for the Poll request -->
1425
1426 <xs:element name="PollResponse" type="dp:PollResponseType"/>
1427
```

1428 **Figure 22. <prov:PollResponse> — Schema Fragment**

1429 An example message body containing a <ps:PollResponse> message follows. This is a successful response
 1430 without an embedded request (and therefore there were no queued requests) and ProvS is advising the PMM to poll
 1431 again in 10 minutes (600 seconds).

```
1432 <prov:PollResponse nextPoll="600">
1433 <lu:Status code="OK" />
1434 </prov:PollResponse>
1435
```

1436 **Example 22. Example <prov:PollResponse> Message**

1437 Another example message body containing a <prov:PollResponse> message follows. This is a successful
 1438 response with an embedded pmm:UpdatePM request.

```
1439 <prov:PollResponse>
1440 <lu:Status code="OK" />
1441 <dp:Request itemID="1">
1442 <pmm:PMUpdate>
1443 <pmm:PMUpdateItem itemID="1" type="urn:liberty:prov:2006-12:ut:engine">
1444 <prov:PMDescriptor xs:id="2323923900239" >
1445 <prov:PMID issuer="http://provs-r-us.com">uuid:778349-2 83920-88379-5448739</prov:PMID>
1446 <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
1447 <ds:Signature>
1448 ... signature data goes here ...
1449 </ds:Signature>
1450 </prov:PMDescriptor>
1451 </pmm:PMUpdateItem>
1452 </pmm:PMUpdate>
1453 </dp:Request>
1454 </prov:PollResponse>
1455
```

1456 **Example 23. Example <prov:PollResponse> Message**

1457 4.13.4. Poll Processing Rules

1458 • All of the processing rules defined for the *Poll* design pattern MUST be followed. See [LibertyDP] for further
 1459 information.

1460 **4.14. Operation: *UpdateEPR***

1461 The *UpdateEPR* operation is used by the PMM to update the `<prov:CallbackEPR>` used by the ProvS to contact
1462 the PMM.

1463 **4.14.1. `wsa:Action` values for *UpdateEPR* Messages**

1464 `<UpdateEPR>` messages MUST include a `<wsa:Action>` SOAP header with the value of "urn:liberty:prov:2006-12:UpdateEPR".

1465 `<UpdateEPRResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of
1466 "urn:liberty:prov:2006-12:UpdateEPRResponse".

1467 **4.14.2. *UpdateEPR* Message**

1468 The `<UpdateEPR>` request is called to update the `<prov:CallbackEPR>` that was registered when the PMM used
1469 the `<prov:PMArtifactResolve>` interface to obtain a `PMDescriptor`.

1470 The `<prov:UpdateEPR>` request contains one or more `<prov:UpdateEPRItem>` elements which have the
1471 following contents:

- 1472 • `<PMID>` **[Required]** - the identifier of the PM that this update is related to. This identifier is included in the
1473 `<prov:PMDescriptor>` returned in the `<prov:PMArtifactResolveResponse>`.
- 1474 • `<prov:CallbackEPR>` **[Required]** - one or more updated ID-WSF EPR(s). This is a complete replacement of
1475 any formerly registered EPRs related to this PM.
- 1476 • `itemID` **[Required]** - the identifier for this request item (for correlation within the results).

1477 The schema for the `<prov:UpdateEPR>` is shown below.

```

1478 <!-- UpdateEPR - update the EPR for PM maintenance operations -->
1479
1480 <xs:element name="UpdateEPR" type="UpdateEPRType"/>
1481
1482 <xs:complexType name="UpdateEPRType">
1483   <xs:complexContent>
1484     <xs:extension base="RequestAbstractType">
1485       <xs:sequence>
1486         <xs:element ref="UpdateEPRItem" maxOccurs="unbounded" />
1487       </xs:sequence>
1488     </xs:extension>
1489   </xs:complexContent>
1490 </xs:complexType>
1491
1492 <xs:element name="UpdateEPRItem" type="UpdateEPRItemType" />
1493
1494 <xs:complexType name="UpdateEPRItemType">
1495   <xs:sequence>
1496     <xs:element ref="PMID" />
1497     <xs:element ref="CallbackEPR" />
1498   </xs:sequence>
1499   <xs:attribute name="itemID" type="xs:string" use="required" />
1500 </xs:complexType>
1501

```

1502 **Figure 23. `<prov:UpdateEPR>` — Schema Fragment**

1503 An example message body containing a `<prov:UpdateEPR>` message follows. This is an update of the `CallbackEPR`
1504 for two different PMs.

```

1505 <prov:UpdateEPR>
1506   <prov:UpdateEPRItem itemID="1" >
1507     <prov:PMID issuer="http://provs-r-us.com">uuid:778349-283920-88379-5448739</prov:PMID>
1508     <prov:CallbackEPR>
1509       <wsa:Address>http://im-a-pmm.com</wsa:Address>
1510       <wsa:Metadata>
1511         <ds:ServiceType>urn:liberty:pmm:2006-12</ds:ServiceType>
1512         <ds:Framework version="2.0" />
1513         <ds:SecurityContext>
1514           <ds:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</ds:SecurityMechID>
1515         </ds:SecurityContext>
1516       </wsa:Metadata>
1517     </prov:CallbackEPR>
1518   </prov:UpdateEPRItem>
1519   <prov:UpdateEPRItem itemID="2">
1520     <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
1521     <prov:CallbackEPR>
1522       <wsa:Address>http://im-a-pmm.com</wsa:Address>
1523       <wsa:Metadata>
1524         <ds:ServiceType>urn:liberty:pmm:2006-12</ds:ServiceType>
1525         <ds:Framework version="2.0" />
1526         <ds:SecurityContext>
1527           <ds:SecurityMechID>urn:liberty:security:2006-08:TLS:SAMLV2</ds:SecurityMechID>
1528         </ds:SecurityContext>
1529       </wsa:Metadata>
1530     </prov:CallbackEPR>
1531   </prov:UpdateEPRItem>
1532 </prov:Update>
1533

```

1534 **Example 24. Example <prov:UpdateEPR> Message**

1535 4.14.3. UpdateEPRResponse Message

1536 This response to the <prov:UpdateEPR> request contains the following elements/attributes:

1537 <lu:Status>: **[Required]** The status of the response. See the processing rules below for more information.

1538 anyAttribute **[Optional]** Zero or more attributes from a namespace other than that of this specification. One
1539 such possibility is an **xs:ID** type attribute such as `xml:id` or `wsu:Id`.

```

1540 <!-- UpdateEPRResponse - the response to the UpdateEPR request -->
1541
1542 <xs:element name="UpdateEPRResponse" type="UpdateEPRResponseType"/>
1543
1544 <xs:complexType name="UpdateEPRResponseType">
1545   <xs:complexContent>
1546     <xs:extension base="ResponseAbstractType" />
1547   </xs:complexContent>
1548 </xs:complexType>
1549

```

1550 **Figure 24. <prov:UpdateEPRResponse> — Schema Fragment**

1551 An example message body containing a <UpdateEPRResponse> message follows. This is a partial success message
1552 for an attempted update of the CallbackEPR for 2 PMs where the 1st succeeded and the 2nd failed.

```

1553 <prov:UpdateEPRResponse>
1554   <lu:Status code="Partial">
1555     <lu:Status code="NotFound" ref="2" />
1556   </lu:Status>
1557 </prov:UpdateEPRResponse>
1558

```

1559 **Example 25. Example `<prov:UpdateEPRResponse>` Message**

1560 **4.14.4. UpdateEPR Processing Rules**

- 1561 • If for any reason an update is not accepted, the existing service instance at the ProvS **MUST NOT** be affected.
- 1562 • PMs instantiated in one PMM **MUST NOT** be visible to other PMMs. If a PMM presents a PMID for a PM
1563 instance that has been instantiated in a different PMM, the ProvS **MUST** behave as if that PM does not exist and
1564 accordingly any attempt to update the associated CallbackEPR should fail. In such cases, if a secondary error
1565 code is provided, it **MUST** be *NotFound*.
- 1566 • The ProvS **SHOULD** generate an error if the CallbackEPR is such that the ProvS is unable to meet the requirements
1567 within the ID-WSF EPR and therefore the ProvS would be unable to dereference the ID-WSF EPR (for example,
1568 if the ProvS does not support the framework version(s) specified in the `<prov:CallbackEPR>`. If a secondary
1569 error code is provided in such cases it **MUST** be *FeatureNotSupported*.
- 1570 • Each `<prov:UpdateEPRItem>` is processed independently and may independently succeed or fail on its own
1571 merits.
- 1572 • If all `<prov:UpdateEPRItem>` requests are successfully processed, the top-level status code **MUST** be *OK*.
1573 If all of the items failed, the the top-level status code **MUST** be *Failed*. Otherwise, if the results were mixed,
1574 the top-level status **MUST** be *Partial* and the second level status **MUST** be included for items for which the
1575 processing was not successful indicating so and including the `ref` attribute containing the `itemID` value for the
1576 item. These second-level status codes **MAY** simply be *Failed*, or they may indicate with more detail the reason
1577 for the failure.
- 1578 • If the top-level status code is *Failed*, the response **MAY** also contain other status codes (such as *NotFound* or
1579 *FeatureNotSupported*) as a second-level status code. The ProvS instance may not wish to reveal the reason for
1580 failure, in which case no second-level status code will appear.

1581 **5. Security and Privacy Considerations**

1582 This section describes security and privacy related considerations that should be taken into account when implementing
1583 and deploying environments making use of the components specified in this document.

1584 The order of entries in this list has no significance.

1585 1. consideration goes here

1586 A. Provisioning Service Schema

```
1587 <?xml version="1.0" encoding="UTF-8"?>
1588 <xs:schema targetNamespace="urn:liberty:prov:2006-12"
1589   xmlns:lu="urn:liberty:util:2006-08"
1590   xmlns:prov="urn:liberty:prov:2006-12"
1591   xmlns:dp="urn:liberty:dp:2006-12"
1592   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1593   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
1594   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1595   xmlns:wsa="http://www.w3.org/2005/08/addressing"
1596   xmlns="urn:liberty:prov:2006-12"
1597   elementFormDefault="qualified"
1598   attributeFormDefault="unqualified"
1599 >
1600
1601 <xs:import namespace="urn:liberty:util:2006-08"
1602   schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>
1603
1604 <xs:import namespace="urn:liberty:dp:2006-12"
1605   schemaLocation="liberty-idwsf-dp-v1.0.xsd"/>
1606
1607 <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
1608   schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.
1609 xsd"/>
1610
1611 <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
1612   schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
1613 <xs:import namespace="http://www.w3.org/2005/08/addressing"
1614   schemaLocation="http://www.w3.org/2005/08/addressing/ws-addr.xsd" />
1615
1616 <xs:annotation>
1617   <xs:documentation>
1618     XML Schema from Liberty Provisioning Service Specification.
1619   </xs:documentation>
1620   <xs:documentation>### NOTICE ###
1621
1622     Copyright (c) 2006 Liberty Alliance participants, see
1623     http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
1624
1625   </xs:documentation>
1626 </xs:annotation>
1627
1628 <!-- PMID - the Provisioned Module Identifier -->
1629
1630 <xs:element name="PMID" type="PMIDType" />
1631 <xs:complexType name="PMIDType">
1632   <xs:attribute name="issuer" type="xs:anyURI" use="required" />
1633 </xs:complexType>
1634
1635
1636 <!-- PMDescriptor - describes/carries the components of a PM -->
1637
1638 <xs:element name="PMDescriptor" type="PMDescriptorType" />
1639
1640 <xs:complexType name="PMDescriptorType">
1641   <xs:sequence>
1642     <xs:element ref="PMID" />
1643     <xs:element ref="PMEngineRef" minOccurs="0" />
1644     <xs:element ref="PMInitData" minOccurs="0" />
1645     <xs:element ref="PMRTData" minOccurs="0" />
1646     <xs:element ref="ds:Signature" minOccurs="0" />
1647   </xs:sequence>
1648   <xs:attribute name="activate" type="xs:boolean" use="optional" />
1649   <xs:attribute name="activateAt" type="xs:dateTime" use="optional" />
1650   <xs:anyAttribute namespace="##other" processContents="lax" />
1651 </xs:complexType>
```

```

1652
1653 <xs:element name="PMEngineRef" type="xs:anyURI" />
1654 <xs:element name="PMInitData" type="EncryptedElementType" />
1655 <xs:element name="PMRTData" type="EncryptedElementType" />
1656
1657 <xs:complexType name="EncryptedElementType">
1658   <xs:sequence>
1659     <xs:element ref="xenc:EncryptedData"/>
1660     <xs:element ref="xenc:EncryptedKey" minOccurs="0" maxOccurs="unbounded"/>
1661   </xs:sequence>
1662 </xs:complexType>
1663
1664
1665 <!-- ProvisioningHandle - Info for PMM to initiate provisioning process -->
1666
1667 <xs:element name="ProvisioningHandle" type="ProvisioningHandleType" />
1668 <xs:complexType name="ProvisioningHandleType">
1669   <xs:sequence>
1670     <xs:element ref="PMArtifact" />
1671     <xs:element ref="ProvisioningServiceEPR" minOccurs="0" />
1672     <xs:element ref="ds:Signature" minOccurs="0" />
1673   </xs:sequence>
1674   <xs:anyAttribute namespace="##other" processContents="lax" />
1675 </xs:complexType>
1676
1677 <xs:element name="ProvisioningServiceEPR" type="wsa:EndpointReferenceType" />
1678
1679 <xs:element name="PMArtifact" type="xs:string" />
1680
1681
1682 <!-- CallbackEPR - where the PMM can receive Provisionig update requests -->
1683
1684 <xs:element name="CallbackEPR" type="wsa:EndpointReferenceType"/>
1685
1686
1687 <!-- PMStatus - Provisioning status of the PM -->
1688
1689 <xs:element name="PMStatus" type="PMStatusType" />
1690 <xs:complexType name="PMStatusType">
1691   <xs:sequence>
1692     <xs:element ref="PMID" />
1693     <xs:element ref="State" />
1694   </xs:sequence>
1695 </xs:complexType>
1696
1697 <xs:element name="State" type="StateType" />
1698 <xs:complexType name="StateType">
1699   <xs:simpleContent>
1700     <xs:extension base="xs:anyURI">
1701       <xs:attribute name="asof" type="xs:dateTime" use="optional" />
1702     </xs:extension>
1703   </xs:simpleContent>
1704 </xs:complexType>
1705
1706
1707 <!-- RequestAbstractType - common request message structure -->
1708
1709 <xs:complexType name="RequestAbstractType" abstract="true">
1710   <xs:anyAttribute namespace="##other" processContents="lax"/>
1711 </xs:complexType>
1712
1713
1714 <!-- ResponseAbstractType - common message response structure -->
1715
1716 <xs:complexType name="ResponseAbstractType" abstract="true">
1717   <xs:sequence>
1718     <xs:element ref="lu:Status"/>

```

```

1719     </xs:sequence>
1720     <xs:anyAttribute namespace="##other" processContents="lax"/>
1721 </xs:complexType>
1722
1723
1724 <!-- GetPMEngine - retrieve the specified PMEEngine -->
1725
1726 <xs:element name="GetPMEngine" type="GetPMEngineType"/>
1727
1728 <xs:complexType name="GetPMEngineType">
1729     <xs:complexContent>
1730         <xs:extension base="RequestAbstractType">
1731             <xs:sequence>
1732                 <xs:element ref="PMEngineRef" />
1733             </xs:sequence>
1734             <xs:attributeGroup ref="dp:BasicPagingAttributeGroup" />
1735         </xs:extension>
1736     </xs:complexContent>
1737 </xs:complexType>
1738
1739 <!-- GetPMEngineResponse - response for the GetPMEngine Request -->
1740
1741 <xs:element name="GetPMEngineResponse" type="GetPMEngineResponseType"/>
1742
1743 <xs:complexType name="GetPMEngineResponseType">
1744     <xs:complexContent>
1745         <xs:extension base="RequestAbstractType">
1746             <xs:sequence>
1747                 <xs:element ref="EngineData" minOccurs="0" />
1748             </xs:sequence>
1749             <xs:attributeGroup ref="dp:BasicPagingResponseAttributeGroup" />
1750         </xs:extension>
1751     </xs:complexContent>
1752 </xs:complexType>
1753
1754 <xs:element name="EngineData" type="xs:base64Binary" />
1755
1756
1757 <!-- PMArtifactResolve - request to exchange PMHandle for PMDescriptor -->
1758
1759 <xs:element name="PMArtifactResolve" type="PMArtifactResolveType"/>
1760
1761 <xs:complexType name="PMArtifactResolveType">
1762     <xs:complexContent>
1763         <xs:extension base="RequestAbstractType">
1764             <xs:sequence>
1765                 <xs:element ref="PMArtifact" />
1766                 <xs:element ref="CallbackEPR" />
1767             </xs:sequence>
1768         </xs:extension>
1769     </xs:complexContent>
1770 </xs:complexType>
1771
1772 <!-- PMArtifactResolveResponse - response to the PMArtifactResolve request -->
1773
1774 <xs:element name="PMArtifactResolveResponse" type="PMArtifactResolveResponse Type"/>
1775
1776 <xs:complexType name="PMArtifactResolveResponseType">
1777     <xs:complexContent>
1778         <xs:extension base="ResponseAbstractType">
1779             <xs:sequence>
1780                 <xs:element ref="PMDescriptor" minOccurs="0"/>
1781             </xs:sequence>
1782         </xs:extension>
1783     </xs:complexContent>
1784 </xs:complexType>
1785
    
```

```
1786 </xs:complexType>
1787
1788
1789 <!-- PMDActivate - to activate one or more PM(s) at the PMM -->
1790
1791 <xs:element name="PMDActivate" type="PMDActivateType"/>
1792
1793 <xs:complexType name="PMDActivateType">
1794   <xs:complexContent>
1795     <xs:extension base="RequestAbstractType">
1796       <xs:sequence>
1797         <xs:element ref="PMDActivateItem" maxOccurs="unbounded" />
1798       </xs:sequence>
1799     </xs:extension>
1800   </xs:complexContent>
1801 </xs:complexType>
1802
1803 <xs:element name="PMDActivateItem" type="PMDActivateItemType" />
1804
1805 <xs:complexType name="PMDActivateItemType">
1806   <xs:sequence>
1807     <xs:element ref="prov:PMID" />
1808   </xs:sequence>
1809   <xs:attribute name="at" type="xs:dateTime" use="optional" />
1810 </xs:complexType>
1811
1812
1813 <!-- PMDActivateResponse - the response to the PMDActivate request -->
1814
1815 <xs:element name="PMDActivateResponse" type="PMDActivateResponseType"/>
1816
1817 <xs:complexType name="PMDActivateResponseType">
1818   <xs:complexContent>
1819     <xs:extension base="ResponseAbstractType" />
1820   </xs:complexContent>
1821 </xs:complexType>
1822
1823
1824 <!-- PMDDeactivate - to deactivate a PM at the PMM -->
1825
1826 <xs:element name="PMDDeactivate" type="PMDDeactivateType"/>
1827
1828 <xs:complexType name="PMDDeactivateType">
1829   <xs:complexContent>
1830     <xs:extension base="RequestAbstractType">
1831       <xs:sequence>
1832         <xs:element ref="PMDDeactivateItem" maxOccurs="unbounded" />
1833       </xs:sequence>
1834     </xs:extension>
1835   </xs:complexContent>
1836 </xs:complexType>
1837
1838 <xs:element name="PMDDeactivateItem" type="PMDDeactivateItemType" />
1839
1840 <xs:complexType name="PMDDeactivateItemType">
1841   <xs:sequence>
1842     <xs:element ref="prov:PMID" />
1843   </xs:sequence>
1844   <xs:attribute name="at" type="xs:dateTime" use="optional" />
1845 </xs:complexType>
1846
1847
1848 <!-- PMDDeactivateResponse - the response to the PMDDeactivate request -->
1849
1850 <xs:element name="PMDDeactivateResponse" type="PMDDeactivateResponseType"/>
1851
1852 <xs:complexType name="PMDDeactivateResponseType">
```

```
1853     <xs:complexContent>
1854         <xs:extension base="ResponseAbstractType" />
1855     </xs:complexContent>
1856 </xs:complexType>
1857
1858
1859 <!-- PMDRegister - to register a new PMD at the ProvS -->
1860
1861 <xs:element name="PMDRegister" type="PMDRegisterType"/>
1862
1863 <xs:complexType name="PMDRegisterType">
1864     <xs:complexContent>
1865         <xs:extension base="RequestAbstractType">
1866             <xs:sequence>
1867                 <xs:element ref="PMDRegisterItem" maxOccurs="unbounded" />
1868             </xs:sequence>
1869         </xs:extension>
1870     </xs:complexContent>
1871 </xs:complexType>
1872
1873 <xs:element name="PMDRegisterItem" type="PMDRegisterItemType" />
1874
1875 <xs:complexType name="PMDRegisterItemType">
1876     <xs:sequence>
1877         <xs:element ref="PMDDescriptor" />
1878     </xs:sequence>
1879     <xs:attribute name="itemID" type="xs:string" use="required" />
1880 </xs:complexType>
1881
1882 <!-- PMDRegisterResponse - response to the PMDRegister request -->
1883
1884 <xs:element name="PMDRegisterResponse" type="PMDRegisterResponseType"/>
1885
1886 <xs:complexType name="PMDRegisterResponseType">
1887     <xs:complexContent>
1888         <xs:extension base="ResponseAbstractType">
1889             <xs:sequence>
1890                 <xs:element ref="PMDRegisterResponseItem" minOccurs="0"/>
1891             </xs:sequence>
1892         </xs:extension>
1893     </xs:complexContent>
1894 </xs:complexType>
1895
1896 <xs:element name="PMDRegisterResponseItem"
1897     type="PMDRegisterResponseItemType"/>
1898
1899 <xs:complexType name="PMDRegisterResponseItemType">
1900     <xs:sequence>
1901         <xs:element ref="ProvisioningHandle" />
1902     </xs:sequence>
1903     <xs:attribute name="ref" type="xs:string" use="required" />
1904 </xs:complexType>
1905
1906
1907 <!-- PMDUpdate - update the PMD for a existing PM at the ProvS -->
1908
1909 <xs:element name="PMDUpdate" type="PMDUpdateType"/>
1910
1911 <xs:complexType name="PMDUpdateType">
1912     <xs:complexContent>
1913         <xs:extension base="RequestAbstractType">
1914             <xs:sequence>
1915                 <xs:element ref="PMDUpdateItem" maxOccurs="unbounded" />
1916             </xs:sequence>
1917         </xs:extension>
1918     </xs:complexContent>
1919 </xs:complexType>
```

```

1920 </xs:complexType>
1921
1922 <xs:element name="PMDUpdateItem" type="PMDUpdateItemType" />
1923
1924 <xs:complexType name="PMDUpdateItemType">
1925   <xs:sequence>
1926     <xs:element ref="PMDDescriptor" />
1927   </xs:sequence>
1928   <xs:attribute name="type" type="xs:anyURI" use="required" />
1929   <xs:attribute name="itemID" type="xs:string" use="required" />
1930 </xs:complexType>
1931
1932 <!-- PMDUpdateResponse - response to the PMDUpdate request -->
1933
1934 <xs:element name="PMDUpdateResponse" type="PMDUpdateResponseType"/>
1935
1936 <xs:complexType name="PMDUpdateResponseType">
1937   <xs:complexContent>
1938     <xs:extension base="ResponseAbstractType">
1939       <xs:sequence>
1940       </xs:sequence>
1941     </xs:extension>
1942   </xs:complexContent>
1943 </xs:complexType>
1944
1945 <!-- PMDDelete - to register a new PMD at the ProvS -->
1946
1947 <xs:element name="PMDDelete" type="PMDDeleteType"/>
1948
1949 <xs:complexType name="PMDDeleteType">
1950   <xs:complexContent>
1951     <xs:extension base="RequestAbstractType">
1952       <xs:sequence>
1953         <xs:element ref="PMID" maxOccurs="unbounded" />
1954       </xs:sequence>
1955     </xs:extension>
1956   </xs:complexContent>
1957 </xs:complexType>
1958
1959 <!-- PMDDeleteResponse - the response to the PMDDelete request -->
1960
1961 <xs:element name="PMDDeleteResponse" type="PMDDeleteResponseType"/>
1962
1963 <xs:complexType name="PMDDeleteResponseType">
1964   <xs:complexContent>
1965     <xs:extension base="ResponseAbstractType" />
1966   </xs:complexContent>
1967 </xs:complexType>
1968
1969 <!-- PMDStatus - to get the status of a PMD at the ProvS -->
1970
1971 <xs:element name="PMDGetStatus" type="PMDGetStatusType"/>
1972
1973 <xs:complexType name="PMDGetStatusType">
1974   <xs:complexContent>
1975     <xs:extension base="RequestAbstractType">
1976       <xs:sequence>
1977         <xs:element ref="PMID" maxOccurs="unbounded" />
1978       </xs:sequence>
1979     </xs:extension>
1980   </xs:complexContent>
1981 </xs:complexType>
1982
1983 <!-- PMDUpdateResponse - response to the PMDUpdate request -->
1984
1985 <xs:element name="PMDUpdateResponse" type="PMDUpdateResponseType"/>
1986

```

```
1987 <!-- PMDStatusResponse - response to the PMDStatus request -->
1988
1989 <xs:element name="PMDStatusResponse" type="PMDStatusResponseType"/>
1990
1991 <xs:complexType name="PMDStatusResponseType">
1992   <xs:complexContent>
1993     <xs:extension base="ResponseAbstractType">
1994       <xs:sequence>
1995         <xs:element ref="PMStatus" minOccurs="0"/>
1996       </xs:sequence>
1997     </xs:extension>
1998   </xs:complexContent>
1999 </xs:complexType>
2000
2001 <!-- PMDSetStatus - update provisioning status of PM -->
2002
2003
2004 <xs:element name="PMDSetStatus" type="PMDSetStatusType"/>
2005
2006 <xs:complexType name="PMDSetStatusType">
2007   <xs:complexContent>
2008     <xs:extension base="RequestAbstractType">
2009       <xs:sequence>
2010         <xs:element ref="PMStatus" />
2011       </xs:sequence>
2012     </xs:extension>
2013   </xs:complexContent>
2014 </xs:complexType>
2015
2016 <!-- PMDSetStatusResonse - response for PMDSetStatus request -->
2017
2018
2019 <xs:element name="PMDSetStatusResponse" type="PMDSetStatusResponseType"/>
2020
2021 <xs:complexType name="PMDSetStatusResponseType">
2022   <xs:complexContent>
2023     <xs:extension base="ResponseAbstractType" />
2024   </xs:complexContent>
2025 </xs:complexType>
2026
2027 <!-- Poll - Poll for new service requests -->
2028
2029
2030 <xs:element name="Poll" type="dp:PollType"/>
2031
2032 <!-- PollResponse - response for the Poll request -->
2033
2034
2035 <xs:element name="PollResponse" type="dp:PollResponseType"/>
2036
2037 <!-- UpdateEPR - update the EPR for PM maintenance operations -->
2038
2039
2040 <xs:element name="UpdateEPR" type="UpdateEPRTYPE"/>
2041
2042 <xs:complexType name="UpdateEPRTYPE">
2043   <xs:complexContent>
2044     <xs:extension base="RequestAbstractType">
2045       <xs:sequence>
2046         <xs:element ref="UpdateEPRItem" maxOccurs="unbounded" />
2047       </xs:sequence>
2048     </xs:extension>
2049   </xs:complexContent>
2050 </xs:complexType>
2051
2052 <xs:element name="UpdateEPRItem" type="UpdateEPRItemTYPE" />
2053
```

```
2054 <xs:complexType name="UpdateEPRItemType">
2055   <xs:sequence>
2056     <xs:element ref="PMID" />
2057     <xs:element ref="CallbackEPR" />
2058   </xs:sequence>
2059   <xs:attribute name="itemID" type="xs:string" use="required" />
2060 </xs:complexType>
2061
2062
2063 <!-- UpdateEPRResponse - the response to the UpdateEPR request -->
2064
2065 <xs:element name="UpdateEPRResponse" type="UpdateEPRResponseType"/>
2066
2067 <xs:complexType name="UpdateEPRResponseType">
2068   <xs:complexContent>
2069     <xs:extension base="ResponseAbstractType" />
2070   </xs:complexContent>
2071 </xs:complexType>
2072
2073
2074 </xs:schema>
2075
```

2076 B. Provisioning Service WSDL

```
2077 <?xml version="1.0"?>
2078 <definitions name="prov-svc"
2079   targetNamespace="urn:liberty:prov:2006-12"
2080   xmlns:tns="urn:liberty:prov:2006-12"
2081   xmlns="http://schemas.xmlsoap.org/wsdl/"
2082   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2083   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2084   xmlns:sb="urn:liberty:sb:2006-12"
2085   xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
2086   xmlns:prov="urn:liberty:prov:2006-12"
2087   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2088   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
2089     http://schemas.xmlsoap.org/wsdl/
2090     http://www.w3.org/2006/02/addressing/wsdl
2091     http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">
2092
2093   <!-- Abstract WSDL for Liberty Provisioning Service v1.0 Specification -->
2094
2095   <xsd:documentation>
2096
2097     Abstract WSDL from Liberty Provisioning Service Specification.
2098
2099     ### NOTICE ###
2100
2101     Copyright (c) 2006 Liberty Alliance participants, see
2102     http://www.projectliberty.org/specs/idwsf\_2\_0\_final\_copyrights.php
2103
2104   </xsd:documentation>
2105
2106   <types>
2107     <xsd:schema>
2108       <xsd:import namespace="urn:liberty:prov:2006-12"
2109         schemaLocation="liberty-idwsf-prov-v1.0.xsd"/>
2110     </xsd:schema>
2111   </types>
2112
2113   <message name="GetPMEngine">
2114     <part name="body" element="prov:GetPMEngine"/>
2115   </message>
2116   <message name="GetPMEngineResponse">
2117     <part name="body" element="prov:GetPMEngineResponse"/>
2118   </message>
2119
2120   <message name="PMArtifactResolve">
2121     <part name="body" element="prov:PMArtifactResolve"/>
2122   </message>
2123   <message name="PMArtifactResolveResponse">
2124     <part name="body" element="prov:PMArtifactResolveResponse"/>
2125   </message>
2126
2127   <message name="PMDActivate">
2128     <part name="body" element="prov:PMDActivate"/>
2129   </message>
2130   <message name="PMDActivateResponse">
2131     <part name="body" element="prov:PMDActivateResponse"/>
2132   </message>
2133
2134   <message name="PMDDeactivate">
2135     <part name="body" element="prov:PMDDeactivate"/>
2136   </message>
2137   <message name="PMDDeactivateResponse">
2138     <part name="body" element="prov:PMDDeactivateResponse"/>
2139   </message>
2140
2141   <message name="PMDDelete">
```

```

2142     <part name="body" element="prov:PMDelete"/>
2143 </message>
2144 <message name="PMDeleteResponse">
2145     <part name="body" element="prov:PMDeleteResponse"/>
2146 </message>
2147
2148 <message name="PMGetStatus">
2149     <part name="body" element="prov:PMGetStatus"/>
2150 </message>
2151 <message name="PMGetStatusResponse">
2152     <part name="body" element="prov:PMGetStatusResponse"/>
2153 </message>
2154
2155 <message name="PMRegister">
2156     <part name="body" element="prov:PMRegister"/>
2157 </message>
2158 <message name="PMRegisterResponse">
2159     <part name="body" element="prov:PMRegisterResponse"/>
2160 </message>
2161
2162 <message name="PMSetStatus">
2163     <part name="body" element="prov:PMSetStatus"/>
2164 </message>
2165 <message name="PMSetStatusResponse">
2166     <part name="body" element="prov:PMSetStatusResponse"/>
2167 </message>
2168
2169 <message name="PMUpdate">
2170     <part name="body" element="prov:PMUpdate"/>
2171 </message>
2172 <message name="PMUpdateResponse">
2173     <part name="body" element="prov:PMUpdateResponse"/>
2174 </message>
2175
2176 <message name="Poll">
2177     <part name="body" element="prov:Poll"/>
2178 </message>
2179 <message name="PollResponse">
2180     <part name="body" element="prov:PollResponse"/>
2181 </message>
2182
2183 <message name="UpdateEPR">
2184     <part name="body" element="prov:UpdateEPR"/>
2185 </message>
2186 <message name="UpdateEPRResponse">
2187     <part name="body" element="prov:UpdateEPRResponse"/>
2188 </message>
2189
2190 <portType name="ProvisioningPort">
2191
2192     <operation name="GetPMEngine">
2193         <input message="tns:GetPMEngine"
2194             wsaw:Action="urn:liberty:prov:2006-12:GetPMEngine" />
2195         <output message="tns:GetPMEngineResponse"
2196             wsaw:Action="urn:liberty:prov:2006-12:GetPMEngineResponse" />
2197     </operation>
2198
2199     <operation name="PMArtifactResolve">
2200         <input message="tns:PMArtifactResolve"
2201             wsaw:Action="urn:liberty:prov:2006-12:PMArtifactResolve" />
2202         <output message="tns:PMArtifactResolveResponse"
2203             wsaw:Action="urn:liberty:prov:2006-12:PMArtifactResolveResponse" />
2204     </operation>
2205
2206     <operation name="PMDActivate">
2207         <input message="tns:PMDActivate"
2208             wsaw:Action="urn:liberty:prov:2006-12:PMDActivate" />
    
```

```

2209     <output message="tns:PMDActivateResponse"
2210         wsaw:Action="urn:liberty:prov:2006-12:PMDActivateResponse" />
2211 </operation>
2212
2213 <operation name="PMDDeactivate">
2214     <input message="tns:PMDDeactivate"
2215         wsaw:Action="urn:liberty:prov:2006-12:PMDDeactivate" />
2216     <output message="tns:PMDDeactivateResponse"
2217         wsaw:Action="urn:liberty:prov:2006-12:PMDDeactivateResponse" />
2218 </operation>
2219
2220 <operation name="PMDDelete">
2221     <input message="tns:PMDDelete"
2222         wsaw:Action="urn:liberty:prov:2006-12:PMDDelete" />
2223     <output message="tns:PMDDeleteResponse"
2224         wsaw:Action="urn:liberty:prov:2006-12:PMDDeleteResponse" />
2225 </operation>
2226
2227 <operation name="PMDGetStatus">
2228     <input message="tns:PMDGetStatus"
2229         wsaw:Action="urn:liberty:prov:2006-12:PMDGetStatus" />
2230     <output message="tns:PMDGetStatusResponse"
2231         wsaw:Action="urn:liberty:prov:2006-12:PMDGetStatusResponse" />
2232 </operation>
2233
2234 <operation name="PMDRegister">
2235     <input message="tns:PMDRegister"
2236         wsaw:Action="urn:liberty:prov:2006-12:PMDRegister" />
2237     <output message="tns:PMDRegisterResponse"
2238         wsaw:Action="urn:liberty:prov:2006-12:PMDRegisterResponse" />
2239 </operation>
2240
2241 <operation name="PMDSetStatus">
2242     <input message="tns:PMDSetStatus"
2243         wsaw:Action="urn:liberty:prov:2006-12:PMDSetStatus" />
2244     <output message="tns:PMDSetStatusResponse"
2245         wsaw:Action="urn:liberty:prov:2006-12:PMDSetStatusResponse" />
2246 </operation>
2247
2248 <operation name="PMDUpdate">
2249     <input message="tns:PMDUpdate"
2250         wsaw:Action="urn:liberty:prov:2006-12:PMDUpdate" />
2251     <output message="tns:PMDUpdateResponse"
2252         wsaw:Action="urn:liberty:prov:2006-12:PMDUpdateResponse" />
2253 </operation>
2254
2255 <operation name="Poll">
2256     <input message="tns:Poll"
2257         wsaw:Action="urn:liberty:prov:2006-12:Poll" />
2258     <output message="tns:PollResponse"
2259         wsaw:Action="urn:liberty:prov:2006-12:PollResponse" />
2260 </operation>
2261
2262 <operation name="UpdateEPR">
2263     <input message="tns:UpdateEPR"
2264         wsaw:Action="urn:liberty:prov:2006-12:UpdateEPR" />
2265     <output message="tns:UpdateEPRResponse"
2266         wsaw:Action="urn:liberty:prov:2006-12:UpdateEPRResponse" />
2267 </operation>
2268
2269 </portType>
2270
2271 <!--
2272 An example of a binding and service that can be used with this
2273 abstract service description is provided below.
2274 -->
2275

```

```

2276 <binding name="ProvisioningBinding" type="tns:ProvisioningPort">
2277
2278   <soap:binding style="document"
2279     transport="http://schemas.xmlsoap.org/soap/http"/>
2280
2281   <operation name="GetPMEngine">
2282     <input> <soap:body use="literal"/> </input>
2283     <output> <soap:body use="literal"/> </output>
2284   </operation>
2285
2286   <operation name="PMArtifactResolve">
2287     <input> <soap:body use="literal"/> </input>
2288     <output> <soap:body use="literal"/> </output>
2289   </operation>
2290
2291   <operation name="PMDActivate">
2292     <input> <soap:body use="literal"/> </input>
2293     <output> <soap:body use="literal"/> </output>
2294   </operation>
2295
2296   <operation name="PMDDeactivate">
2297     <input> <soap:body use="literal"/> </input>
2298     <output> <soap:body use="literal"/> </output>
2299   </operation>
2300
2301   <operation name="PMDDelete">
2302     <input> <soap:body use="literal"/> </input>
2303     <output> <soap:body use="literal"/> </output>
2304   </operation>
2305
2306   <operation name="PMDGetStatus">
2307     <input> <soap:body use="literal"/> </input>
2308     <output> <soap:body use="literal"/> </output>
2309   </operation>
2310
2311   <operation name="PMDRegister">
2312     <input> <soap:body use="literal"/> </input>
2313     <output> <soap:body use="literal"/> </output>
2314   </operation>
2315
2316   <operation name="PMDSetStatus">
2317     <input> <soap:body use="literal"/> </input>
2318     <output> <soap:body use="literal"/> </output>
2319   </operation>
2320
2321   <operation name="PMDUpdate">
2322     <input> <soap:body use="literal"/> </input>
2323     <output> <soap:body use="literal"/> </output>
2324   </operation>
2325
2326   <operation name="Poll">
2327     <input> <soap:body use="literal"/> </input>
2328     <output> <soap:body use="literal"/> </output>
2329   </operation>
2330
2331   <operation name="UpdateEPR">
2332     <input> <soap:body use="literal"/> </input>
2333     <output> <soap:body use="literal"/> </output>
2334   </operation>
2335
2336 </binding>
2337
2338 <service name="ProvisioningService">
2339
2340   <port name="ProvisioningPort" binding="tns:ProvisioningBinding">
2341
2342     <!-- Modify with the REAL SOAP endpoint -->

```

```
2343
2344     <soap:address location="http://example.com/provisioning"/>
2345
2346     </port>
2347
2348     </service>
2349
2350 </definitions>
2351
```

2352 References

2353 Normative

- 2354 [LibertyPMM] Cahill, Conor, eds. "Liberty ID-WSF Provisioned Module Manager Service Specification," Version
2355 1.0, Liberty Alliance Project (15 December, 2006). <http://www.projectliberty.org/specs>
- 2356 [LibertyPROV] Cahill, Conor, eds. "Liberty ID-WSF Provisioning Service Specification," Version 1.0, Liberty
2357 Alliance Project (15 December, 2006). <http://www.projectliberty.org/specs>
- 2358 [LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-
2359 errata-1.0-01, Liberty Alliance Project (29 November, 2006). <http://www.projectliberty.org/specs>
- 2360 [LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification,"
2361 Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 2362 [LibertyDP] Cahill, Conor, eds. "Liberty ID-WSF Design Patterns Specification," Version 1.0, Liberty Alliance Project
2363 (15 December, 2006). <http://www.projectliberty.org/specs>
- 2364 [LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0,
2365 Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>
- 2366 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version
2367 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 2368 [LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14
2369 December, 2004). <http://www.projectliberty.org/specs>
- 2370 [LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. "Liberty ID-WSF Authentication,
2371 Single Sign-On, and Identity Mapping Services Specification," v2.0-errata-1.0-01, Liberty Alliance Project
2372 (28 November, 2006). <http://www.projectliberty.org/specs>
- 2373 [LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Lib-
2374 erty ID-WSF SOAP Binding Specification," Version 2.0, Liberty Alliance Project (30 July, 2006).
2375 <http://www.projectliberty.org/specs>
- 2376 [RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
2377 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt>
- 2378 [RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251,
2379 Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2251.txt>
- 2380 [RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A.
2381 Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force
2382 <http://www.ietf.org/rfc/rfc2252.txt>
- 2383 [RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search
2384 Results," RFC 2891, Internet Engineering Task Force <http://www.ietf.org/rfc/rfc2891.txt>
- 2385 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Pro-
2386 tocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS
2387 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-
2388 open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf)

- 2389 [SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions
2390 and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OA-
2391 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-
2392 open.org/security/saml/v2.0/saml-core-2.0-os.pdf)
- 2393 [SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler,
2394 Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,"
2395 SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards
2396 <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- 2397 [Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October
2398 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium
2399 <http://www.w3.org/TR/xmlschema-1/>
- 2400 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman,
2401 Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C
2402 Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 2403 [XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004).
2404 "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium
2405 <http://www.w3.org/TR/2004/REC-xml-20040204>
- 2406 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
2407 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 2408 [XPath] Clark, J., DeRose, S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0,"
2409 Recommendation, W3C <http://www.w3.org/TR/xpath> [August 2003].

2410 Informative

- 2411 [LibertyACT] Cahill, Conor P., eds. "Liberty Advanced Client Technologies," Version 1.0-17, Liberty Alliance Project
2412 (15 January, 2007). <http://www.projectliberty.org/specs>
- 2413 [LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty
2414 Alliance Project (13 January, 2005). <http://www.projectliberty.org/specs>
- 2415 [LibertyIDPPGuide] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Implementation
2416 Guidelines," Version 1.1, Liberty Alliance Project (29 September, 2005). <http://www.projectliberty.org/specs>
- 2417 [LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework
2418 Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). <http://www.projectliberty.org/specs>