Liberty ID-WSF Provisioning Service Specification

Version: 1.0

Editors:
Conor P. Cahill, Intel Corporation

Contributors:
Hubert Le Van Gong, Sun
Paul Madsen, NTT
Hiroyoshi Takiguchi, NTT
Greg Whitehead, HP
Janice Yang, Intel Corporation

Abstract:

This specification defines the Provisioning Service interfaces.

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

Notice

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

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

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

Contents

1. Introduction

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

1.1. Notation and Conventions

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

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

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

1.1.1. XML Namespaces

The following XML namespaces are referred to in this document:

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

  urn:liberty:prov:2007-09

- The prefix *pmm:* stands for the Liberty ID-WSF Provisioned Module Manager namespace [LibertyPMM]:

  urn:liberty:pmm:2007-09

- The prefix *saml2:* stands for the SAMLv2 assertion namespace [SAMLCore2]:

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

- The prefix *samlp2:* stands for the SAMLv2 protocol namespace [SAMLCore2]:

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

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

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

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

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

2. Overview

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

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

2.1. Provisioning Components

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

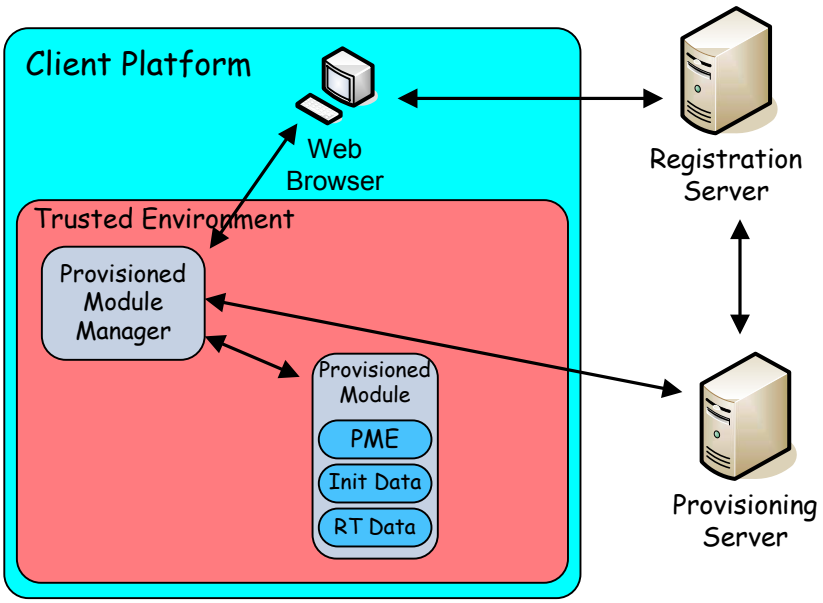

Figure 1. Provisioning Components

Things to note about this diagram:

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

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

- The Provisioned Module (PM) is a component which performs some set of functionality. For example, a PM could be a TM (a module which provides IdP extension functionality). PMs may also expose functionality that is not defined by Liberty specifications.

  Each PM is identified using a globally unique identifier called the Provisioned Module IDentifier (PMID). The PMID is used to reference to specific instances of a PM when performing tasks like status updates or module updates.

  The PM is shown as being composed of 3 distinct parts:

  - **Provisioned Module Engine (PME)** - the executable code which provides the functionality for the PM.

    This is defined as a separate component here to enable a provisioning process which allows the PME to preexist in the client platform and so just delivers the data necessary to instantiate the PM using that preexisting engine. Of course, the PME may not preexist and in such cases the PMM will have to retrieve it.

    During provisioning, the PME is passed by reference (name) so that the PMM can determine whether or not the PME already exists (either because it was pre-installed or because the same PME has been previously provisioned). Should the PMM need to obtain the PME, the passed in reference is used to identify the PME being downloaded.

  - **Initialization Data (PMInitData)** - the data needed by the PME in order to initialize a new instance of a PM. This may be the actual data needed by the PME or it may be a reference that the PME knows how to dereference and obtain the initialization data at runtime. This data may or may not be needed during the provisioning process. Some PMs are fully individualized and have their PMInitData built in.

    The format and structure of the PMInitData is out of scope for this document and is specific to the PME. It is up to the Provisioning Service to resolve what data is needed for what PME. The PMM treats PMInitData as an opaque data set that it passes to the PME upon initialization.

  - **Runtime Data (PMRTData)** - the runtime data created/managed by the PM instance as it performs its tasks. This would include things like MINGs for a TM that is minting assertions, private keys, etc. This is defined separate from the InitData to allow for PM portability (where a previously activated PM is moved to another client platform).

- The Web Browser in this diagram represents an enhanced browser (either directly or via a plug-in) with support for the provisioning process. In other provisioning use cases this may be an application or even be the PMM itself can instigate a new provision operation (typically via some direct interaction with the user).

- The Registration Server (RegS) is not a Liberty defined entity, but rather a deployment component for a particular set of use cases. In this use case, the RegS interacts with the user through a web browser and then controls the provisioning process using the interfaces on the Provisioning Server.

- The Provisioning Server (ProvS) is typically a network hosted service that is the primary entity with which the PMM interacts. This server is an instance of a Liberty ID-WSF Provisioning Service (see [LibertyPROV]).

  The primary function of the ProvS is to provide a trusted endpoint for the management and distribution of PMs.

3. Data Definitions

3.1. Provisioned Module Identifier

A <prov:PMID> is a unique identifier assigned to the PM when the PMD is first registered with the ProvS (i.e., before the PM has been provisioned). This identifier is used on all subsequent communications with the ProvS relating to that PM from the registering party as well as from the PMM. Therefore the value selected for the PMID MUST be unique across all such values issued by that issuing authority.

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

- issuer **[Required]** - the Provider ID of the party which issued the PMID. This is used to prevent collision of PMID values for PMs issued by different parties.

The following schema fragment defines the **<prov:PMID>** element:

```
<!-- PMID - the Provisioned Module IDentifier  -->

<xs:element name="PMID" type="PMIDType" />
<xs:complexType name="PMIDType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:attribute name="issuer" type="xs:anyURI" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

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

```
<prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
```

3.2. ProvisioningHandle

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

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

- <prov:PMDArtifact> **[Required]** - a token issued by the ProvS which identifies a specific PM that is to be provisioned to the Advanced client.

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

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

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

- `<prov:ProvisioningServiceEPR>` **[optional]** - zero or more ID-WSF EPR(s) describing the endpoint for the entity that is provisioning this PM.

  Normally, the EPR(s) are specified within the PH. However, there MAY be some environments where the PMM has preexisting knowledge of the ProvS EPR and so this element is listed as optional to allow it to be not specified in such cases.

- `<ds:Signature>` **[Optional]** - an XML digital signature ([XMLDsig]) covering the entire descriptor.

  If a signature is present, the signer MUST follow the normative rules laid out in Section 5 ("SAML and XML Signature Syntax and Processing") of the SAML 2.0 Core Specification ([SAMLCore2]).

- `expires` **[Optional]** - an optional attribute containing the time at which the data within this handle is deemed to be no longer usable. Artifacts presented after the expiration of this time will not be resolvable.

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

The following schema fragment defines the **`<prov:ProvisioningHandle>`** element:

```
<!-- ProvisioningHandle - Info for PMM to initiate provisioning process -->

<xs:element name="ProvisioningHandle" type="ProvisioningHandleType"/>
<xs:complexType name="ProvisioningHandleType">
  <xs:sequence>
    <xs:element ref="PMDArtifact"  />
    <xs:element ref="ProvisioningServiceEPR" minOccurs="0"
                                maxOccurs="unbounded" />
    <xs:element ref="ds:Signature"        minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

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

<xs:element name="PMDArtifact" type="xs:string" />
```

The following is an example of a **`<prov:ProvisioningHandle>`** element.

```
<prov:ProvisioningHandle xs:id="2302384823023">
  <prov:PMDArtifact>23asdfhoi323hposdf923h9sdfhweorh2398asdfjweoiha</prov:PMDArtifact>
  <prov:ProvisioningServiceEPR>
    <wsa:Address>http://provision.idpsRus.com</wsa:Address>
    <wsa:Metadata>
      <ds:Abstract>Provisioning Service</ds:Abstract>
      <ds:ProviderID>http://provisioning-provider.idpsRus.com/</ds:ProviderID>
      <ds:ServiceType>urn:liberty:prov:2007-09</ds:ServiceType>
      <ds:Framework version="2.0" />
      <ds:SecurityContext>
        <ds:SecurityMechID>
          urn:liberty:security:2005-02:TLS:SAMLV2
        </ds:SecurityMechID>
        <sec:Token ref="urn:liberty:disco:tokenref:ObtainFromIDP" />
      </ds:SecurityContext>
    </wsa:Metadata>
  </prov:ProvisioningServiceEPR>
  <ds:Signature>
    ... signature info here ..
  </ds:Signature>
</prov:ProvisioningHandle>
```

3.3. PMDescriptor

The <prov:PMDescriptor>, or Provisioned Module Descriptor (PMD), contains the information necessary for the PMM to instantiate a PM within the Advanced Client.

The <prov:PMDescriptor> is typically acquired by the PMM following the dereference of the <PMDArtifact> at the ProvS.

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

- <ds:Signature> **[Optional]** - an XML digital signature ([XMLDsig]) covering the entire descriptor.

  If a signature is present, the signer MUST follow the normative rules laid out in Section 5 ("SAML and XML Signature Syntax and Processing") of the SAML 2.0 Core Specification ([SAMLCore2]).

- <prov:PMID> **[Required]** - the Provisioned Module IDentifier for the PM described by this descriptor.

- <prov:PMEngineRef> **[Optional]** - a URI reference to a PMEngine. This engine may already exist on the Advanced Client, or it may need to be downloaded and installed by the PMM. This element MUST be present in any <prov:PMDescriptor> that is describing a complete PM. However, the element MAY be absent when the <prov:PMDescriptor> is used to describe an update.

- <prov:PMEngine> **[Optional]** - the complete description and contents of the PM engine. Normally, due to size and processing constraints, the engine itself is not included in the <prov:PMDescriptor>. However, when that isn't the case, the executable can be included in this element. If this element is present, the <prov:PMEngineRef> element MUST NOT be specified.

  This element contains the following sub-elements:

  - <PMEInfo> **[Required]** - the information about the engine. For a complete description of the contents of this element see see Section 3.5.

  - <PMEBytes> **[Required]** - the base64 encoded bytes of the engine (the actual bits of the executable code for the engine).

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

- <prov:PMInitData> **[Optional]** - initialization data needed by the PMEngine to initialize as a PM instance. The structure and content of this element are defined by, and specific to, the PM being provisioned. In many cases, this will contain an encrypted element containing the initialization data, but many other solutions could be used including a base64 element, an artifact, etc.

- <prov:PMRTData> **[Optional]** - runtime data saved by a previous instance of the PM associated with this descriptor. Like the <prov:PMInitData> element, the content and structure of this element is defined by, and specific to the PM being provisioned.

- activate **[optional]** - a boolean attribute indicating whether the PM should be activated upon installation. If this attribute is not specified or it its value is *true*, the PM is activated. Otherwise the PM is installed, but not activated.

  See the related activateAt and deactivateAt attributes below for automated future activation/deactivation.

- activateAt **[optional]** - an attribute indicating when the PM is to be activated. The value of this attribute is a specific instance in time at which the PM should be activated (functionality enabled).

  This is typically used when a PM is to be installed now for a future activation.

  This attribute MUST ONLY be specified when the activate attribute's value is *true* (explicitly or implicitly).

- `deactivateAt` **[optional]** - an attribute indicating when the PM is to be deactivated. The value of this attribute is a specific instance in time at which the PM should be deactivated.

  This is typically used when a PM is to be installed now for a future activation.

  This attribute MUST ONLY be specified when the `activate` attribute's value is *true* (explicitly or implicitly).

  If the `activateAt` attribute is also specified, the value of the `deactivateAt` attribute value MUST NOT be the same instant as, or earlier than, the `activateAt` attribute value.

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

The following schema fragment defines the **`<prov:PMDescriptor>`** element:

```
<!-- PMDescriptor - describes/carries the components of a PM -->

<xs:element name="PMDescriptor" type="PMDescriptorType"/>

<xs:complexType name="PMDescriptorType">
  <xs:sequence>
    <xs:element ref="ds:Signature" minOccurs="0"/>
    <xs:element ref="PMID"                />
    <xs:element ref="PMEngineRef"  minOccurs="0"/>
    <xs:element ref="PMEngine"    minOccurs="0"/>
    <xs:element ref="PMInitData"   minOccurs="0"/>
    <xs:element ref="PMRTData"    minOccurs="0"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="activate"    type="xs:boolean"  use="optional"/>
  <xs:attribute name="activateAt"  type="xs:dateTime" use="optional"/>
  <xs:attribute name="deactivateAt" type="xs:dateTime" use="optional"/>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>

<xs:element name="PMInitData"  type="PMDataType" />
<xs:element name="PMRTData"    type="PMDataType" />

<xs:complexType name="PMDataType" mixed="false">
  <xs:sequence>
    <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="PMEngine"    type="PMEngineType" />

<xs:complexType name="PMEngineType" mixed="false">
  <xs:sequence>
    <xs:element ref="PMEInfo"  />
    <xs:element ref="PMEBytes" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="PMEBytes"    type="xs:base64Binary" />
```

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

```
<prov:PMDescriptor xs:id="2323923900239" >
  <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
  <prov:PMInitData>
    <MyData>
```

```
      .... initialization data here ...
    </MyData>
  </prov:PMInitData>
  <ds:Signature>
    ... signature data goes here ...
  </ds:Signature>
</prov:PMDescriptor>
```

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

```
<prov:PMDescriptor xs:id="2323923900239"
     activateAt="2007-01-11T14:52:00Z" >
  <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
  <prov:PMEngineRef>http://pmsRus.org/ConorsPM/1.0</prov: PMEngineRef>
  <ds:Signature>
    ... signature data goes here ...
  </ds:Signature>
</prov:PMDescriptor>
```

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

```
<prov:PMDescriptor xs:id="2323923900239"
     activate="false" >
  <prov:PMID issuer="http://provs-r-us.com">uuid: 239032-230328-92379-2397923</ prov:PMID>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
  <prov:PMInitData>
    <MyData>
      .... initialization data here ...
    </MyData>
  </prov:PMInitData>
  <ds:Signature>
    ... signature data goes here ...
  </ds:Signature>
</prov:PMDescriptor>
```

3.4. PMStatus

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

The <prov:PMStatus> element has the following attributes:

- <prov:PMID> **[Required]** - the PMID for the PM that this status applies to.
- <prov:State> **[Required]** - the current state of the provisioning of this PM. This element contains the URI value representing the current state as well as the following attribute:
  - asof **[Optional]** - the times when the current status of the PM was achieved. This attribute SHOULD NOT be specified when setting a new status at the ProvS or the PMM and SHOULD ONLY be present on status retrievals (e.g., <PMGetStatus>).

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

- *urn:liberty:prov:2007-09:status:Registered* - the <PMDescriptor> has been registered with the ProvS and the ProvS is awaiting the <PMGetDescriptor> request from the PMM.

- *urn:liberty:prov:2007-09:status:Resolved* - the PMM has obtained the <PMDescriptor> via the <prov:PMGetDescriptor> request and the ProvS is awaiting a status update from the PMM to indicate the provisioning is complete.

- *urn:liberty:prov:2007-09:status:Active* - the PM is installed and active.

- *urn:liberty:prov:2007-09:status:Inactive* - the PM is installed, but is currently inactive.

- *urn:liberty:prov:2007-09:status:Deleted* - the PM has been deleted. The actual existence of this status is implementation-defined. Some implementations will immediately remove the PMD data from their database upon deletion and thus never expose this status as the PMD would not exist. Other implementations will maintain the PMD in the database for some implementation-defined period of time with this deleted status.

  Implementations of ProvS clients SHOULD NOT depend upon the existence of this status, nor upon the immediate removal of a PMD upon deletion (e.g., don't expect to be able to reuse the PMID of a deleted PMD although some implementations MAY allow this).

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

```
<!-- PMStatus - Provisioning status of the PM -->

<xs:element name="PMStatus" type="PMStatusType"/>
<xs:complexType name="PMStatusType">
  <xs:sequence>
    <xs:element ref="PMID"/>
    <xs:element ref="State"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="State" type="StateType"/>
<xs:complexType name="StateType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:attribute name="asof"  type="xs:dateTime" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

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

```
<prov:PMStatus>
  <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
  <prov:State asof="2007-01-14T17:31:11Z">urn:liberty:prov:2007-09:status:Active</prov:State>
</prov:PMStatus>
```

3.5. PMEInfo

The <prov:PMEInfo>, (or Provisioned Module Engine Information) describes a provisioned module engine (the executable code portion of a PM).

The <prov:PMEInfo> element has the following attributes:

- <prov:PMEngineRef> **[Required]** - the identity token (or name of) the provisioned module engine. Each version of each engine MUST have a unique URI string to identify the exact engine.

- `<prov:PMECreatorID>` **[Required]** - a URI containing the provider ID of the entity which registered the PMEngine at the ProvS.
- `<prov:PMEWhenCreated>` **[Required]** - the moment in time when the PMEngine was registered (created) at the ProvS.
- `<prov:PMEEnabled>` **[Required]** - a boolean flag indicating whether or not this PMEngine is enabled for downloading by a PMM.
- `<prov:PMEWhenEnabled>` **[Required]** - the moment in time when the enabled status of the PMEngine was last changed.
- `<prov:PMESize>` **[Required]** - the size, in bytes, of the PMEngine.
- `<prov:PMEHash>` **[Required]** - a hash digest of the PMEngine. This element contains the hash value representing the current state as well as the following attribute:
  - `method` **[Optional]** - the algorithm used to generate the hash digest. The possible values are taken from the XML Signature Syntax and Processing, which currently only lists the SHA-1 hash URI:
    - *http://www.w3.org/2000/09/xmldsig#sha1*

The following schema fragment defines the **`<prov:PMEInfo>`** element:

```
<!-- PMEInfo - The current state of a PMEngine -->

<xs:element name="PMEInfo" type="PMEInfoType"/>
<xs:complexType name="PMEInfoType">
  <xs:sequence>
    <xs:element ref="PMEngineRef"/>
    <xs:element ref="PMECreatorID"/>
    <xs:element ref="PMEWhenCreated"/>
    <xs:element ref="PMEEnabled"/>
    <xs:element ref="PMEWhenEnabled"/>
    <xs:element ref="PMESize"/>
    <xs:element ref="PMEHash"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax" />
</xs:complexType>

<xs:element name="PMEngineRef" type="xs:anyURI"/>

<xs:element name="PMECreatorID" type="xs:anyURI"/>

<xs:element name="PMEWhenCreated" type="xs:dateTime"/>

<xs:element name="PMEEnabled" type="xs:boolean"/>

<xs:element name="PMEWhenEnabled" type="xs:dateTime"/>

<xs:element name="PMESize" type="xs:integer"/>

<xs:element name="PMEHash" type="PMEHashType"/>
<xs:complexType name="PMEHashType">
  <xs:simpleContent>
    <xs:extension base="xs:base64Binary">
      <xs:attribute name="method" type="xs:anyURI" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

The following is an example of a **<prov:PMEInfo>** element. In this case, the provisioning of the PM is complete.

```
<prov:PMEInfo>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.6</prov:PMEngineRef>
  <prov:PMECreatorID>http://reg.providers.com</prov:PMECreatorID>
  <prov:PMEWhenCreated>2007-01-18T17:32:14Z</prov:PMEWhenCreated>
  <prov:PMEEnabled>true</prov:PMEEnabled>
  <prov:PMEWhenEnabled>2007-01-18T18:15:22Z</prov:PMEWhenEnabled>
  <prov:PMESize>185676</prov:PMESize>
  <prov:PMEHash method="http://www.w3.org/2000/09/xmldsig#sha1">
    ...SHA1 hash data ...
  </prov:PMEHash>
</prov:PMEInfo>
```

3.6. Callback EPR

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

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

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

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

```
<!-- CallbackEPR - where the PMM can receive Provisionig update requests -->

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

Figure 2. <prov:CallbackEPR> — Schema Fragment

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

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

Example 1. Example anonymous <prov:CallbackEPR>

4. Provisioning Service (ProvS)

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

An abstract WSDL definition for the Provisioning Service is included in this document, see Section 6: Provisioning Service WSDL . This WSDL document defines all of the "WSDL operations" for the IdP Service.

The complete schema for the Provisioning Service is included in this document, see Section 5: Provisioning Service Schema .

4.1. Service URIs

Table 1. ProvS Service URIs

Use	URI
Service Type	*urn:liberty:prov:2007-09*
PMActivate wsa:Action	*urn:liberty:prov:2007-09:PMActivate*
PMActivateResponse wsa:Action	*urn:liberty:prov:2007-09:PMActivateResponse*
PMDeactivate wsa:Action	*urn:liberty:prov:2007-09:PMDeactivate*
PMDeactivateResponse wsa:Action	*urn:liberty:prov:2007-09:PMDeactivateResponse*
PMDelete wsa:Action	*urn:liberty:prov:2007-09:PMDelete*
PMDeleteResponse wsa:Action	*urn:liberty:prov:2007-09:PMDeleteResponse*
PMGetDescriptor wsa:Action	*urn:liberty:prov:2007-09:PMGetDescriptor*
PMGetDescriptorResponse wsa:Action	*urn:liberty:prov:2007-09:PMGetDescriptorResponse*
PMGetStatus wsa:Action	*urn:liberty:prov:2007-09:PMGetStatus*
PMGetStatusResponse wsa:Action	*urn:liberty:prov:2007-09:PMGetStatusResponse*
PMRegisterDescriptor wsa:Action	*urn:liberty:prov:2007-09:PMRegisterDescriptor*
PMRegisterDescriptorResponse wsa:Action	*urn:liberty:prov:2007-09:PMRegisterDescriptorResponse*
PMSetStatus wsa:Action	*urn:liberty:prov:2007-09:PMSetStatus*
PMSetStatusResponse wsa:Action	*urn:liberty:prov:2007-09:PMSetStatusResponse*
PMUpdate wsa:Action	*urn:liberty:prov:2007-09:PMUpdate*
PMUpdateResponse wsa:Action	*urn:liberty:prov:2007-09:PMUpdateResponse*
Poll wsa:Action	*urn:liberty:prov:2007-09:Poll*
PollResponse wsa:Action	*urn:liberty:prov:2007-09:PollResponse*
UpdateEPR wsa:Action	*urn:liberty:prov:2007-09:UpdateEPR*
UpdateEPRResponse wsa:Action	*urn:liberty:prov:2007-09:UpdateEPRResponse*

616 **Table 2. ProvS Service URIs, part 2**

Use	URI
PMERegister wsa:Action	*urn:liberty:prov:2007-09:PMERegister*
PMERegisterResponse wsa:Action	*urn:liberty:prov:2007-09:PMERegisterResponse*
PMEDelete wsa:Action	*urn:liberty:prov:2007-09:PMEDelete*
PMEDeleteResponse wsa:Action	*urn:liberty:prov:2007-09:PMEDeleteResponse*
PMEDisable wsa:Action	*urn:liberty:prov:2007-09:PMEDisable*
PMEDisableResponse wsa:Action	*urn:liberty:prov:2007-09:PMEDisableResponse*
PMEDownload wsa:Action	*urn:liberty:prov:2007-09:PMEDownload*
PMEDownloadResponse wsa:Action	*urn:liberty:prov:2007-09:PMEDownloadResponse*
PMEEnable wsa:Action	*urn:liberty:prov:2007-09:PMEEnable*
PMEEnableResponse wsa:Action	*urn:liberty:prov:2007-09:PMEEnableResponse*
PMEGetInfo wsa:Action	*urn:liberty:prov:2007-09:PMEGetInfo*
PMEGetInfoResponse wsa:Action	*urn:liberty:prov:2007-09:PMEGetInfoResponse*
PMEUpload wsa:Action	*urn:liberty:prov:2007-09:PMEUpload*
PMEUploadResponse wsa:Action	*urn:liberty:prov:2007-09:PMEUploadResponse*

617 ## 4.2. Status Codes

618 The following status code strings are defined:

619 • *OK*: message processing succeeded

620 • *Failed*: general failure code

621 • *Forbidden*: action by invoker is forbidden.

622 • *Issued*: the specified artifact has already been issued.

623 • *NotFound*: the requested object is not present.

624 • *Expired*: the presented token/artifact has expired.

625 • *Duplicate*: the object already exists or the task has already been accomplished.

626 • *Invalid*: one or more of the specified parameters are invalid.

627 • *WrongSize*: the engine size is incorrect.

628 • *HashMisMatch*: the engine hash does not match the registered value.

629 • *LimitExceeded*: too much data was sent to the ProvS.

630 • *OutOfOrder*: data was sent in an incorrect order.

631 • *FeatureNotSupported*: the request involves some capability that is not supported.

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

4.3. Request and Response Abstract Types

4.3.1. Complex Type RequestAbstractType

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

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

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

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

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

```

4.3.2. Complex Type ResponseAbstractType

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

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

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

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

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

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

```

4.4. Operation: *PMGetDescriptor*

The *PMGetDescriptor* operation is used by the PMM to exchange a `<PMDArtifact>` for a `<PMDescriptor>`.

4.4.1. `wsa:Action` values for `PMGetDescriptor` Messages

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

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

4.4.2. `PMGetDescriptor` Message

The `<PMGetDescriptor>` request is called to exchange a `<PMDArtifact>` for a `<PMDescriptor>`.

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

- `PMDArtifact` **[Required]** - the PMDArtifact that is being resolved.
- `<prov:CallbackEPR>` **[Required]** - one or more ID-WSF Endpoint References which describe how the ProvS can communicate with the PMM for maintenance operations (see Section 3.6).

  If more than one `<prov:CallbackEPR>` element is present, the ProvS may choose any of the elements present (recognizing that the list is in preference order – the most preferred element is first).
- `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMGetDescriptor - request to exchange PMHandle for PMGetDescriptor -->

<xs:element name="PMGetDescriptor" type="PMGetDescriptorType"/>

<xs:complexType name="PMGetDescriptorType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMDArtifact"/>
        <xs:element ref="CallbackEPR" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 3. `<prov:PMGetDescriptor>` — Schema Fragment

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

```
<prov:PMGetDescriptor>
  <prov:PMDArtifact>23asdfhoi323hpo sdf923h9sdfhweorh2398asdfjweoi ha</prov:PMDArtifact>
  <prov:CallbackEPR>
    <wsa:Address>http://www.w3.org/2005/08/ addressing/anonymous</wsa:Add ress>
  </prov:CallbackEPR>
</prov:PMGetDescriptor>
```

Example 2. Example `<prov:PMGetDescriptor>` Message

4.4.3. `PMGetDescriptorResponse` Message

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

- `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- `<prov:PMDescriptor>` **[Optional]** - the Provisioned Module Descriptor that is associated with the supplied `<prov:PMDArtifact>`. This element will not be present on unsuccessful responses.

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

```
<!-- PMGetDescriptorResponse - response to the PMGetDescriptor request -->

<xs:element name="PMGetDescriptorResponse" type="PMGetDescriptorResponseType"/>

<xs:complexType name="PMGetDescriptorResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMDescriptor" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 4. `<prov:PMGetDescriptorResponse>` — Schema Fragment

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

```
<prov:PMGetDescriptorResponse>
  <lu:Status code="OK" />
  <prov:PMDescriptor xs:id="2323923900239">
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
    <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
    <prov:PMInitData>
      <MyData>
        .... initialization data here ...
      </MyData>
    </prov:PMInitData>
    <ds:Signature>
      ... signature data goes here ...
    </ds:Signature>
  </prov:PMDescriptor>
</prov:PMGetDescriptorResponse>
```

Example 3. Example `<prov:PMGetDescriptorResponse>` Message

4.4.4. PMGetDescriptor Processing Rules

- If the PMDescriptor has already been issued for the specified PMDArtifact (e.g., someone has already invoked this interface to resolve the artifact), the request MUST fail. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Issued*."

- If the artifact is presented after the PH has expired, the ProvS SHOULD refuse to process it and instead return a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Expired*."

- If the artifact is not known to the ProvS, the ProvS MUST treat the request as a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."

- The Provisioning Service SHOULD take steps to ensure that the correct party is submitting the request and to reject requests from inappropriate parties.

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

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

4.5. Operation: *PMActivate*

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

4.5.1. `wsa:Action` values for `PMActivate` Messages

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

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

4.5.2. `PMActivate` Message

The `<PMActivate>` request is called by a provisioning entity (such as the RegS show in the Provisioning workflow in the Liberty ID-WSF Advanced Client Technologies Overview [LibertyACT] to activate an inactive PM.

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

- `<prov:PMActivateItem>` **[Required]** - one or more activate actions which contain the following elements/attributes:

  - `<prov:PMID>` **[Required]** - the identifier of the PM being activated.

  - `itemID` **[Required]** - a unique (within this message) identifier for this request item (used for correlation of result status in the response).

  - `at` **[Optional]** - a time at which the activation should take place. If specified, the value SHOULD be some point in the future at which the PM would be activated.

    If this attribute is not specified, or it is specified with a time in the past, the request is treated as an immediate request.

    This attribute MUST NOT be interpreted or acted upon by the ProvS (with the potential exception of optional scheduling of multiple queued operations). The subsequent `<pmm:PMDeactivate>` request to the PMM MUST include this value and SHOULD be sent as soon as possible (e.g., do NOT wait till the specified time to deliver the message).

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

- `<dp:NotifyTo>` **[Optional]** - an optional endpoint reference for delayed notification completion messages (see [LibertyDP]). This is used to allow the invoker to receive a completion status on the eventual activation of the PM at the PMM.

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

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

```
<!-- PMActivate - to activate one or more PM(s) at the PMM -->

<xs:element name="PMActivate" type="PMActivateType"/>

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

<xs:element name="PMActivateItem" type="PMActivateItemType" />

<xs:complexType name="PMActivateItemType">
  <xs:sequence>
    <xs:element ref="prov:PMID"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string"   use="required"/>
  <xs:attribute name="at"    type="xs:dateTime"  use="optional"/>
</xs:complexType>
```

Figure 5. <prov:PMActivate> — Schema Fragment

An example message body containing a <prov:PMActivate> message follows. This request activates 2 PMs, one at 1PM on the 18th of Dec, 2006, and one now and provides a notification endpoint for delayed notification of the completion status.

```
<prov:PMActivate>
  <prov:PMActivateItem itemID="1" at="2007-01-18T13:00:00Z" >
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
  </prov:PMActivateItem>
  <prov:PMActivateItem itemID="2">
    <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-18923-2389238</prov:PMID>
  </prov:PMActivateItem>
  <dp:NotifyTo>
    <wsa:Address>https://provider.com/notifications</wsa:Address>
    <wsa:Metadata>
      <ds:ProviderID>http://provider.com/</ds:ProviderID>
      <ds:ServiceType>urn:liberty:dp:2007-09:notification</ds:ServiceType>
      <ds:Framework version="2.0" />
      <ds:SecurityContext>
        <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:null</ds:SecurityMechID>
      </ds:SecurityContext>
    </wsa:Metadata>
  </dp:NotifyTo>
</prov:PMActivate>
```

Example 4. Example <prov:PMActivate> Message

4.5.3. PMActivateResponse Message

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

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

```
<!-- PMActivateResponse - the response to the PMActivate request -->

<xs:element name="PMActivateResponse" type="PMActivateResponseType"/>

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

Figure 6. `<prov:PMActivateResponse>` — Schema Fragment

An example message body containing a `<prov:PMActivateResponse>` message follows. This is a partial response. The 2nd item in the request succeeded, but the first item is still in process and a delayed notification will be sent upon completion.

```
<prov:PMActivateResponse>
  <lu:Status code="Partial">
    <lu:Status ref="1" code="WillNotify"/>
    <lu:Status ref="2" code="OK"      />
  </lu:Status>
</prov:PMActivateResponse>
```

Example 5. Example `<prov:PMActivateResponse>` Message

In this second example for the `<prov:PMActivateResponse>` message, the delayed notification message containing the completion status for the first item is returned.

```
<prov:PMActivateResponse>
  <lu:Status code="Partial">
    <lu:Status ref="1" code="OK"/>
  </lu:Status>
</prov:PMActivateResponse>
```

Example 6. Example delayed `<prov:PMActivateResponse>` Message

4.5.4. PMActivate Processing Rules

- This operation adopts the Delayed Notification design pattern (see [LibertyDP]) and incorporates all of the associated processing rules. Delayed notifications are needed for both indirect operations (where the PMM is polling for incoming request) and for delayed operations (where the `at` attribute is used with a time in the future).

  Delayed Notifications SHOULD NOT to be used for immediate operations (no future `at` attribute) targeted at a PMM that has exposed a means of direct communications from the ProvS (e.g., they have registered a non-anonymous CallbackEPR with the ProvS). In such cases, the ProvS SHOULD wait for the completion of the subsequent request to the PMM and return the appropriate status in its response to the invoker.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS NOT present on the request, the ProvS should validate the request to the extent possible and respond with the results of that validation. If that validation succeeded, the ProvS MUST continue with the execution of the operation. In such cases, the actual completion status of the operation may be different from what was reported (e.g., the request may fail at the PMM for some reason), but that will not be known to the invoker, given that they have chosen to not provide a delayed notification endpoint.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS present on the request, the ProvS MUST report the final completion status (e.g., wait for the completion status of any delayed or indirect operation at the PMM) of the request. If multiple operations are included in the request, the ProvS MAY group the results of some or all of the operations in a single delayed notification message.

  In the case of a mixed request, the immediate portion of the request SHOULD be handled as an immediate request and the actual results returned with the initial response.

- Requests to activate a PM that is not known to the ProvS MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this error MUST be "*NotFound*."

- Requests to activate a PM that has not yet been provisioned (i.e., the PMDescriptor has been received, but not provisioned to a PMM) MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this error MUST be "*NotProvisioned*."

- The ProvS SHOULD accept and forward this request regardless of the current status of the PM (provided that it has been provisioned and has not been deleted). The return status of this request should reflect the eventual return status from the PMM.

- If **all** items in the request have the same completion status, the top level status MUST reflect that completion status and MUST be "*OK*," "*Failed*," or "*WillNotify*." Otherwise, if the results were mixed, the top-level status MUST be *Partial* and a second-level status MUST be included indicating which items succeeded, which failed, and which will be subject to delayed notification. The second level status elements MUST include the `ref` attribute containing the `itemID` value for the item. For failures, the second-level status codes MAY simply be "*Failed*" or they may indicate, with more detail, the reason for the failure.

  If the top-level status is "*Failed*," second level status codes MAY be present which contain detailed error information if the ProvS wants to share that information with the invoking party.

4.6. Operation: *PMDeactivate*

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

4.6.1. `wsa:Action` values for `PMDeactivate` Messages

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

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

4.6.2. `PMDeactivate` Message

The <PMDeactivate> request is called by a provisioning entity (such as the RegS show in the Provisioning workflow in the Liberty ID-WSF Advanced Client Technologies Overview [LibertyACT] to deactivate an activated PM.

The <prov:PMDeactivate> request contains the following elements/attributes:

- <prov:PMDeactivateItem> **[Required]** - one or more deactivation actions which contain the following elements/attributes:
  - <PMID> **[Required]** - the identifier of the PM being deactivated.
  - itemID **[Required]** - a unique (within this message) identifier for this request item (used for correlation of result status in the response).
  - at **[Optional]** - a time at which the deactivation should take place. If specified, the value SHOULD be some point in the future at which the PM would be deactivated.

    If this attribute is not specified, or it is specified with a time in the past, it is interpreted as an immediate request.

    This attribute MUST NOT be interpreted or acted upon by the ProvS (with the potential exception of optional scheduling of multiple queued operations). The subsequent <pmm:PMDeactivate> request to the PMM MUST include this value and SHOULD be sent as soon as possible (e.g., do NOT wait till the specified time to deliver the message).

    Note that since this request is made to the ProvS and not the PMM, the time it takes to relay the message, especially in a scenario where the PMM is polling the ProvS, may delay the deactivation time, especially when this attribute has a very short time frame.
- <dp:NotifyTo> **[Optional]** - an optional endpoint reference for delayed notification completion messages (see [LibertyDP]). This is used to allow the invoker to receive a completion status on the eventual activation of the PM at the PMM.
- anyAttribute **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **xs:ID** type attribute such as xml:id or wsu:Id.

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

```
<!-- PMDeactivate - to deactivate a PM at the PMM -->

<xs:element name="PMDeactivate" type="PMDeactivateType"/>

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

<xs:element name="PMDeactivateItem" type="PMDeactivateItemType" />

<xs:complexType name="PMDeactivateItemType">
  <xs:sequence>
    <xs:element ref="prov:PMID"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string"   use="required"/>
  <xs:attribute name="at"    type="xs:dateTime"  use="optional"/>
</xs:complexType>

```

Figure 7. `<prov:PMDeactivate>` — Schema Fragment

An example message body containing a `<prov:PMDeactivate>` message follows. This request deactivates two PMs, one now and one near midnight on New Year's Eve.

```
<prov:PMDeactivate>
  <prov:PMDeactivateItem itemID="1" at="2007-01-31T23:59:59Z" >
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-2 30328-92379-2397923</prov:PMI D>
  </prov:PMDeactivateItem>
  <prov:PMDeactivateItem itemID="2">
    <prov:PMID issuer="http://provs-r-us.com">uui d:392380-948492-18923-2389238 </prov:PMID>
  </prov:PMDeactivateItem>
</prov:PMDeactivate>
```

Example 7. Example `<prov:PMDeactivate>` Message

4.6.3. `PMDeactivateResponse` Message

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

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

```
<!-- PMDeactivateResponse - the response to the PMDeactivate request -->

<xs:element name="PMDeactivateResponse" type="PMDeactivateResponseType" />

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

Figure 8. `<prov:PMDeactivateResponse>` — Schema Fragment

An example message body containing a `<prov:PMDeactivateResponse>` message follows. This is a partially successful response where one of the PMs was not found.

```
<prov:PMDeactivateResponse>
  <lu:Status code="Partial">
    <lu:Status ref="1" code="NotFound"/>
  </lu:Status>
</prov:PMDeactivateResponse>
```

Example 8. Example `<prov:PMDeactivateResponse>` Message

4.6.4. PMDeactivate Processing Rules

- This operation adopts the Delayed Notification design pattern (see [LibertyDP]) and incorporates all of the associated processing rules. Delayed notifications are needed for both indirect operations (where the PMM is polling for incoming request) and for delayed operations (where the `at` attribute is used with a time in the future).

  Delayed Notifications SHOULD NOT to be used for immediate operations (no future `at` attribute) targeted at a PMM that has exposed a means of direct communications from the ProvS (e.g., they have registered a non-anonymous CallbackEPR with the ProvS). In such cases, the ProvS SHOULD wait for the completion of the subsequent request to the PMM and return the appropriate status in its response to the invoker.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS NOT present on the request, the ProvS should validate the request to the extent possible and respond with the results of that validation. If that validation succeeded, the ProvS MUST continue with the execution of the operation. In such cases, the actual completion status of the operation may be different from what was reported (e.g., the request may fail at the PMM for some reason), but that will not be known to the invoker, given that they have chosen to not provide a delayed notification endpoint.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS present on the request, the ProvS MUST report the final completion status (e.g., wait for the completion status of any delayed or indirect operation at the PMM) of the request. If multiple operations are included in the request, the ProvS MAY group the results of some or all of the operations in a single delayed notification message.

  In the case of a mixed request, the immediate portion of the request SHOULD be handled as an immediate request and the actual results returned with the initial response.

- Requests to deactivate a PM that is not known to the ProvS MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this error MUST be "*NotFound*."

- Requests to deactivate a PM that has not yet been provisioned (i.e., the PMDescriptor has been received, but not provisioned to a PMM) MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this error MUST be "*NotProvisioned*."

- The ProvS SHOULD accept and forward this request regardless of the current state of the PMDescriptor (provided that it does exist and has not been deleted).

- If **all** items in the request have the same completion status, the top level status MUST reflect that completion status and MUST be "*OK*," "*Failed*," or "*WillNotify*." Otherwise, if the results were mixed, the top-level status MUST be *Partial* and a second-level status MUST be included indicating which items succeeded, which failed, and which will be subject to delayed notification. The second level status elements MUST include the `ref` attribute containing the `itemID` value for the item. For failures, the second-level status codes MAY simply be "*Failed*" or they may indicate, with more detail, the reason for the failure.

  If the top-level status is "*Failed*," second level status codes MAY be present which contain detailed error information if the ProvS wants to share that information with the invoking party.

4.7. Operation: *PMDelete*

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

4.7.1. `wsa:Action` values for `PMDelete` Messages

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

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

4.7.2. `PMDelete` Message

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

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

- `<PMDeleteItem>` **[Required]** - one or more request elements containing the following elements or attributes:
  - `<PMID>` **[Required]** - the identifier for the PMDescriptor that is to be deleted. This element MUST contain a value that matches the `PMID` of a previously registered `<PMDescriptor>`.
  - `itemID` **[Required]** - a unique (within this message) identifier for this request item (used for correlation of result status in the response).
- `<dp:NotifyTo>` **[Optional]** - an optional endpoint reference for delayed notification completion messages (see [LibertyDP]). This is used to allow the invoker to receive a completion status on the eventual activation of the PM at the PMM.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMDelete - to register a new PM at the ProvS -->

<xs:element name="PMDelete" type="PMDeleteType"/>
<xs:complexType name="PMDeleteType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMDeleteItem"  maxOccurs="unbounded" />
        <xs:element ref="dp:NotifyTo"   minOccurs="0"      />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMDeleteItem" type="PMDeleteItemType" />
<xs:complexType name="PMDeleteItemType">
  <xs:sequence>
    <xs:element ref="prov:PMID"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string"   use="required"/>
</xs:complexType>
```

Figure 9. `<prov:PMDelete>` — Schema Fragment

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

```
<prov:PMDelete>
  <prov:PMDeleteItem itemID="1">
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
  </prov:PMDeleteItem>
  <prov:PMDeleteItem itemID="2">
    <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-18923-2389238</prov:PMID>
  </prov:PMDeleteItem>
</prov:PMDelete>
```

Example 9. Example `<prov:PMDelete>` Message

4.7.3. `PMDeleteResponse` Message

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

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

```
<!-- PMDeleteResponse - the response to the PMDelete request -->

<xs:element name="PMDeleteResponse" type="PMDeleteResponseType"/>

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

Figure 10. `<prov:PMDeleteResponse>` — Schema Fragment

An example message body containing a `<PMDeleteResponse>` message follows. In this case, the response indicates a partial success and the detailed status codes which indicate that the second PMID was not found.

```
<prov:PMDeleteResponse>
  <lu:Status code="Partial">
    <lu:Status ref="1" code="NotFound"/>
  </lu:Status>
</prov:PMDeleteResponse>
```

Example 10. Example `<prov:PMDeleteResponse>` Message

4.7.4. PMDelete Processing Rules

- This operation adopts the Delayed Notification design pattern (see [LibertyDP]) and incorporates all of the associated processing rules. Delayed notifications are needed for both indirect operations (where the PMM is polling for incoming request) and for delayed operations (where the `at` attribute is used with a time in the future).

  Delayed Notifications SHOULD NOT to be used for immediate operations (no future `at` attribute) targeted at a PMM that has exposed a means of direct communications from the ProvS (e.g., they have registered a non-anonymous CallbackEPR with the ProvS). In such cases, the ProvS SHOULD wait for the completion of the subsequent request to the PMM and return the appropriate status in its response to the invoker.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS NOT present on the request, the ProvS should validate the request to the extent possible and respond with the results of that validation. If that validation succeeded, the ProvS MUST continue with the execution of the operation. In such cases, the actual completion status of the operation may be different from what was reported (e.g., the request may fail at the PMM for some reason), but that will not be known to the invoker, given that they have chosen to not provide a delayed notification endpoint.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS present on the request, the ProvS MUST report the final completion status (e.g., wait for the completion status of any delayed or indirect operation at the PMM) of the request. If multiple operations are included in the request, the ProvS MAY group the results of some or all of the operations in a single delayed notification message.

  In the case of a mixed request, the immediate portion of the request SHOULD be handled as an immediate request and the actual results returned with the initial response.

- Requests to delete a PM that is not known to the ProvS MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this error MUST be "*NotFound*."

- If the PM has been provisioned into a PMM, the ProvS MUST initiate a `<pmm:PMDelete>` operation at the PMM to delete the provisioned PM. This request SHOULD NOT be considered complete until the deletion is completed at the PMM.

- The ProvS SHOULD accept and forward this request regardless of the current state of the PMDescriptor (provided that it does exist and has not already been deleted).

- If **all** items in the request have the same completion status, the top level status MUST reflect that completion status and MUST be "*OK*," "*Failed*," or "*WillNotify*." Otherwise, if the results were mixed, the top-level status MUST be *Partial* and a second-level status MUST be included indicating which items succeeded, which failed, and which will be subject to delayed notification. The second level status elements MUST include the `ref` attribute containing the `itemID` value for the item. For failures, the second-level status codes MAY simply be "*Failed*" or they may indicate, with more detail, the reason for the failure.

  If the top-level status is "*Failed*," second level status codes MAY be present which contain detailed error information if the ProvS wants to share that information with the invoking party.

4.8. Operation: *PMGetStatus*

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

4.8.1. `wsa:Action` values for `PMGetStatus` Messages

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

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

4.8.2. `PMGetStatus` Message

The `<PMGetStatus>` request is called to obtain the status of a PM that was registered with the ProvS.

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

- `<PMID>` **[Optional]** - zero or more identifiers for the PMs who's status is requested. If specified, this element MUST contain a value that matches the `PMID` of a previously registered `<PMDescriptor>`.

  If not specified, the ProvS should return the status for all PMDescriptors registered by the invoker (if any).

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

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

```
<!-- PMStatus - to get the status of a PM at the ProvS -->

<xs:element name="PMGetStatus" type="PMGetStatusType"/>

<xs:complexType name="PMGetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMID"  minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 11. <prov:PMGetStatus> — Schema Fragment

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

```
<prov:PMGetStatus>
  <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-9237 9-2397923</prov:PMID>
  <prov:PMID issuer="http://provs-r-us.com">uuid:392380-948492-1 8923-2389238</prov:PMID>
</prov:PMGetStatus>
```

Example 11. Example <prov:PMGetStatus> Message

4.8.3. PMGetStatusResponse Message

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

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

```
<!-- PMGetStatusResponse - response to the PMGetStatus request -->

<xs:element name="PMGetStatusResponse" type="PMGetStatusResponseType"/>

<xs:complexType name="PMGetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMStatus" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 12. `<prov:PMGetStatusResponse>` — Schema Fragment

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

```
<prov:PMGetStatusResponse>
  <lu:Status code="OK" />
  <prov:PMStatus>
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-23 97923</prov:PMID>
    <prov:State asof="2007-01-14T17:31:11Z">urn:liberty:prov:2007-09:status:Active</prov:State>
  </prov:PMStatus>
</prov:PMGetStatusResponse>
```

Example 12. Example `<prov:PMGetStatusResponse>` Message

4.8.4. PMGetStatus Processing Rules

- The ProvS MUST silently ignore `<PMID>` values of which it is not aware. If all of the `<PMID>` values specified on the request are unknown to the ProvS, the ProvS should return a successful response with no status records.

  In other words, specifying a non-existent `<PMID>` does not result in an error. Instead it results in nothing in the output.

- When this interface is used by a party other than the PMD-issuing authority or the PMM where the PM was provisioned, the ProvS MUST treat the PMD as if the ProvS was not aware of it (i.e., the PMD does not exist as far as other invoking parties are concerned).

- If there are NO PMIDs specified in the request, the ProvS MUST return the current status of all PMs that have been registered by the invoker.

  This rule does NOT apply if PMIDs are specified which refer to nonexistent PMs.

- If request processing succeeded for **any** of the requested PMDs, the top-level status code MUST be "*OK*." If the request processing failed for all PMDs, the top-level status code MUST be "*Failed*." Partial results, where some of the items were found and some were not found, are considered a successful response that only include the `<prov:PMStatus>` element(s) for the PMDs that were found.

- If the top-level status is not *OK*. and second level status codes are present, they MAY contain detailed error information if the ProvS wants to share that information with the invoking party.

4.9. Operation: *PMRegisterDescriptor*

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

4.9.1. `wsa:Action` values for `PMRegisterDescriptor` Messages

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

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

4.9.2. `PMRegisterDescriptor` Message

The `<PMRegisterDescriptor>` request is called to register a new PMD with the ProvS.

The `<prov:PMRegisterDescriptor>` request contains one or more `<prov:PMRegisterDescriptorItem>` elements which have the following contents:

- `<PMDescriptor>` **[Required]** - the PMD that is being registered.
- `itemID` **[Required]** - a unique (within this message) identifier for this request item (used for correlation of result status in the response).

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

```
<!-- PMRegisterDescriptor - to register a new PMD at the ProvS -->

<xs:element name="PMRegisterDescriptor" type="PMRegisterDescriptorType"/>

<xs:complexType name="PMRegisterDescriptorType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMRegisterDescriptorItem"  maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMRegisterDescriptorItem" type="PMRegisterDescriptorItemType" />

<xs:complexType name="PMRegisterDescriptorItemType">
  <xs:sequence>
    <xs:element ref="PMDescriptor" />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>
```

Figure 13. `<prov:PMRegisterDescriptor>` — Schema Fragment

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

```
<prov:PMRegisterDescriptor>
  <prov:PMRegisterDescriptorItem itemID="1">
    <prov:PMDescriptor xs:id="2323923900239">
      <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
      <prov:PMInitData>
        <MyData>
          .... initialization data here ...
        </MyData>
      </prov:PMInitData>
      <ds:Signature>
        ... signature data goes here ...
      </ds:Signature>
    </prov:PMDescriptor>
  </prov:PMRegisterDescriptorItem>
</prov:PMRegisterDescriptor>
```

Example 13. Example `<prov:PMRegisterDescriptor>` Message

4.9.3. `PMRegisterDescriptorResponse` Message

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

- `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- `<prov:PMRegisterDescriptorResponseItem>` **[Optional]** - zero or more response items which are included for successful responses. If present, this element contains the following content:
  - `<prov:ProvisioningHandle>` **[Required]** - the Provisioning Handle created by the ProvS to provide a PMM with the information necessary to obtain the PMD that was registered.
  - `ref` **[Required]** - a reference to the `itemID` of the `<prov:PMRegisterDescriptorItem>` element that this `<prov:PMRegisterDescriptorResponseItem>` is in response to.
- `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- PMRegisterDescriptorResponse - response to the PMRegisterDescriptor request -->

<xs:element name="PMRegisterDescriptorResponse" type="PMRegisterDescriptorResponseType"/>

<xs:complexType name="PMRegisterDescriptorResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMRegisterDescriptorResponseItem" minOccurs="0"
                                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMRegisterDescriptorResponseItem"
        type="PMRegisterDescriptorResponseItemType"/>

<xs:complexType name="PMRegisterDescriptorResponseItemType">
  <xs:sequence>
    <xs:element ref="ProvisioningHandle"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required"/>
</xs:complexType>
```

Figure 14. `<prov:PMRegisterDescriptorResponse>` — Schema Fragment

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

```
<prov:PMRegisterDescriptorResponse>
  <lu:Status code="OK" />
  <prov:PMRegisterDescriptorResponseItem ref="1">
    <prov:ProvisioningHandle xs:id="2302384823023">
      <prov:PMDArtifact>23asdfhoi323hpos df923h9sdfhweorh2398asdfjweoih a</prov:PMDArtifact>
      <prov:ProvisioningServiceEPR>
        <wsa:Address>http://provision.idpsRus.com</wsa:Add ress>
        <wsa:Metadata>
          <ds:Abstract>Provisioning Service</ds:Abstract>
          <ds:ProviderID>http://provisioning-provider.idps Rus.com/</ds:ProviderID>
          <ds:ServiceType>urn:liberty:prov:2007-09</ds:Serv iceType>
          <ds:Framework version="2.0" />
          <ds:SecurityContext>
            <ds:SecurityMechID>
              urn:liberty:security:2005-02:TLS:SAMLV2
            </ds:SecurityMechID>
            <sec:Token ref="urn:liberty:disco:tokenref:ObtainFromIDP" />
          </ds:SecurityContext>
        </wsa:Metadata>
      </prov:ProvisioningServiceEPR>
      <ds:Signature>
        ... signature info here ..
      </ds:Signature>
    </prov:ProvisioningHandle>
  </prov:PMRegisterDescriptorResponseIt em>
</prov:PMRegisterDescriptorResponse>
```

Example 14. Example `<prov:PMRegisterDescriptorResponse>` Message

4.9.4. PMRegisterDescriptor Processing Rules

- The invoking party MUST NOT specify a value for the `<PMID>` element in the `<PMDescriptor>` that duplicates a value previously used. In such cases, the request MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this case MUST be "*Duplicate*."

- The ProvS MAY require that the PMEngine referenced within the `<PMDescriptor>` first be registered with the ProvS before allowing a PMD to be registered. If this is the case, and if an attempt is made to register a `<PMDescriptor>` with an engine that has not been registered, the ProvS MAY treat that as a failure. If this is the case and if detailed status codes are included in the response, the detailed status code for this case MUST be "*UnknownEngine*".

- Each `<prov:PMRegisterDescriptorItem>` is processed independently and may independently succeed or fail on its own merits.

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

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

4.10. Operation: *PMSetStatus*

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

4.10.1. `wsa:Action` values for `PMSetStatus` Messages

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

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

4.10.2. `PMSetStatus` Message

The `<PMSetStatus>` request is called to notify the Provisioning Service about the status of the provisioning of a PM.

The `<prov:PMSetStatus>` request contains following attributes and/or elements:

- `<PMStatus>` - the updated status (see Section 3.4) of the PM. The `<State>` element SHOULD NOT include an `asof` attribute – the ProvS will assign the time itself.
- `anyAttribute` **[Optional]** Zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMSetStatus - update provisioning status of PM -->

<xs:element name="PMSetStatus" type="PMSetStatusType"/>

<xs:complexType name="PMSetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMStatus"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 15. `<prov:PMSetStatus>` — Schema Fragment

An example message body containing a `<PMSetStatus>` message follows. This request shows that the PM has been installed.

```
<prov:PMSetStatus>
  <prov:PMStatus>
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230328-92379-2397923</prov:PMID>
    <prov:State>urn:liberty:prov:2007-09:status:Active</prov:State>
  </prov:PMStatus>
</prov:PMSetStatus>
```

Example 15. Example `<prov:PMSetStatus>` Message

4.10.3. `PMSetStatusResponse` Message

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

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

```
<!-- PMSetStatusResonse - response for PMSetStatus request -->

<xs:element name="PMSetStatusResponse" type="PMSetStatusResponseType"/>

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

Figure 16. `<prov:PMSetStatusResponse>` — Schema Fragment

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

```
<prov:PMSetStatusResponse>
  <lu:Status code="OK" />
</prov:PMSetStatusResponse>
```

Example 16. Example `<prov:PMSetStatusResponse>` Message

4.10.4. PMSetStatus Processing Rules

- If a specified PMID is not known to the ProvS, the ProvS MUST treat that request item as a failure. If detailed status codes are being included in the response, the detailed status code for this error MUST be "*NotFound*."
- The ProvS MUST ensure that only the PMM where the PM is provisioned can use this interface to directly change the status of the PM. Invocations of this interface by any other party MUST result in a failure.

  If the invoker is the PMD-issuing authority, and detailed status codes are being included in the response, the detailed status code for this error MUST be "*Forbidden*." The only way for an issuing authority to change the status of a PMD is indirectly through one of the PMD management interfaces.

  Otherwise, if detailed status codes are being included in the response, the detailed status code for this error MUST be "*NotFound*."
- If the `<State>` element within the `<PMStatus>` contains an `asof` attribute, the ProvS MUST ignore this value and, instead, use its understanding of the current time to mark the status time.
- If the `<State>` element within the `<PMStatus>` contains a value that is not understood by the ProvS, the ProvS MAY treat the operation as a failure. ProvSs MUST accept all of the Liberty-defined status values specified in this document.

  If the request is to be treated as a failure for this reason and detailed status codes are included in the response, the detailed status code for this error MUST be "*Invalid*."
- If request processing succeeded, the top-level status code MUST be "*OK*." Otherwise, the top-level status code MUST be "*Failed*"
- If the top-level status code is "*Failed*," the response MAY also contain *Forbidden* or *NotFound* as a second-level status code. The Provisioning Service instance may not wish to reveal the reason for failure. In which case, no second-level status code will appear.

4.11. Operation: *PMUpdate*

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

4.11.1. `wsa:Action` values for `PMUpdate` Messages

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

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

4.11.2. `PMUpdate` Message

The `<PMUpdate>` request is called to update a PMD that was registered with the ProvS.

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

- `<prov:PMUpdateItem>` **[Required]** - one or more update actions which contain the following elements/attributes:
  - `<PMDescriptor>` **[Required]** - the updated PMD information. The `<PMID>` element MUST contain a value that matches the `<PMID>` of a previously registered `<PMDescriptor>`.

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

    The `activate`, `activateAt`, and `deactivateAt` attributes SHOULD NOT be specified when a `<PMDescriptor>` is used in an update request. The standard interfaces for managing the activation status are the only way to update/change the activation status of a PMD.
  - `type` **[Required]** - the type of update being applied. The following values are defined for this attribute:
    - *urn:liberty:prov:2007-09:ut:replace* - a complete replacement of the existing PMD (and if it has been provisioned, the PM itself).
    - *urn:liberty:prov:2007-09:ut:engine* - an update of the engine, only.
    - *urn:liberty:prov:2007-09:ut:initdata* - an update of the initialization data.
    - *urn:liberty:prov:2007-09:ut:cancel* - the cancellation of a pending future update. If this type is specified, the `at` attribute MUST be specified, MUST contain the exact same value that was used on the update that is being canceled, and the `<PMDescriptor>` element MUST ONLY include the `<PMID>` element corresponding to the PMID used in the pending update.
  - `itemID` **[Required]** - the identifier for this request item (for correlation within the results).

- `at` **[Optional]** - an optional time at which the update should take place. If specified, this SHOULD be some time in the future.

  This attribute MUST NOT be interpreted or acted upon by the ProvS (with the potential exception of optional scheduling of multiple queued operations). The subsequent `<pmm:PMUpdate>` request to the PMM MUST include this value and SHOULD be sent as soon as possible (e.g., do NOT wait till the specified time to deliver the message).

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

- `<dp:NotifyTo>` **[Optional]** - an optional endpoint reference for delayed notification completion messages (see [LibertyDP]). This is used to allow the invoker to receive a completion status on the eventual activation of the PM at the PMM.

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

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

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

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

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

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

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

Figure 17. `<prov:PMUpdate>` — Schema Fragment

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

```
<prov:PMUpdate>
  <prov:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239" >
      <prov:PMID issuer="http://provs-r-us.com">uuid:778349-283920-88379-5448739</prov:PMID>
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
      <ds:Signature>
        ... signature data goes here ...
      </ds:Signature>
    </prov:PMDescriptor>
  </prov:PMUpdateItem>
</prov:PMUpdate>
```

Example 17. Example `<prov:PMUpdate>` Message

4.11.3. `PMUpdateResponse` Message

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

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

```
<!-- PMUpdateResponse - response to the PMUpdate request -->

<xs:element name="PMUpdateResponse" type="PMUpdateResponseType"/>

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

Figure 18. `<prov:PMUpdateResponse>` — Schema Fragment

An example message body containing a `<PMUpdateResponse>` message follows. In this case, the response indicates a failure and the detailed status code indicates the PMID was not found.

```
<prov:PMUpdateResponse>
  <lu:Status code="Failed">
    <lu:Status ref="1" code="NotFound" />
  </lu:Status>
</prov:PMUpdateResponse>
```

Example 18. Example `<prov:PMUpdateResponse>` Message

4.11.4. PMUpdate Processing Rules

- This operation adopts the Delayed Notification design pattern (see [LibertyDP]) and incorporates all of the associated processing rules. Delayed notifications are needed for both indirect operations (where the PMM is polling for incoming request) and for delayed operations (where the `at` attribute is used with a time in the future).

  Delayed Notifications SHOULD NOT to be used for immediate operations (no future `at` attribute) targeted at a PMM that has exposed a means of direct communications from the ProvS (e.g., they have registered a non-anonymous CallbackEPR with the ProvS). In such cases, the ProvS SHOULD wait for the completion of the subsequent request to the PMM and return the appropriate status in its response to the invoker.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS NOT present on the request, the ProvS should validate the request to the extent possible and respond with the results of that validation. If that validation succeeded, the ProvS MUST continue with the execution of the operation. In such cases, the actual completion status of the operation may be different from what was reported (e.g., the request may fail at the PMM for some reason), but that will not be known to the invoker, given that they have chosen to not provide a delayed notification endpoint.

  If any portion of the request IS NOT an immediate operation and the `<dp:NotifyTo>` element IS present on the request, the ProvS MUST report the final completion status (e.g., wait for the completion status of any delayed or indirect operation at the PMM) of the request. If multiple operations are included in the request, the ProvS MAY group the results of some or all of the operations in a single delayed notification message.

  In the case of a mixed request, the immediate portion of the request SHOULD be handled as an immediate request and the actual results returned with the initial response.

- The invoking party MUST specify a value for the `<PMID>` element in the `<PMDescriptor>` that matches a value previously used **by this issuing party**. If this is not the case, the ProvS MUST reject the update request and return a failure status code. If detailed status codes are included in that message, the detailed status code MUST be "*NotFound*."

- If a cancellation is received for a pending update that has NOT been transmitted to the PMM yet, the ProvS SHOULD remove the pending update without transmitting it to the PMM. If the invoker of the update has asked for delayed notification, the notification message for the canceled update MUST include a failure status code. If detailed status codes are included in the delayed notification message, the detailed status code MUST be "*Canceled*." In such case, the response to the cancellation request itself SHOULD indicate success.

- If a cancellation is received for an update that is not currently pending (already completed, never seen, etc.), the request MUST result in a failure. If detailed status codes are included in the delayed notification message, the detailed status code MUST be "*NotFound*."

  Update requests that have been forwarded to the PMM, but which are not known by the ProvS to have been completed, are considered to be pending and the cancellation request MUST be forwarded to the PMM.

- If an update request includes any `activate`, `activateAt`, or `deactivateAt` attributes within the `<PMDescriptor>`, these attributes SHOULD be ignored. The update request cannot be used to change the activation status of a PMD.

- Each `<prov:PMUpdateItem>` is processed independently and may independently succeed or fail on its own merits.

- If **all** items in the request have the same completion status, the top level status MUST reflect that completion status and MUST be "*OK*," "*Failed*," or "*WillNotify*." Otherwise, if the results were mixed, the top-level status MUST be *Partial* and a second-level status MUST be included indicating which items succeeded, which failed, and which will be subject to delayed notification. The second level status elements MUST include the `ref` attribute containing the `itemID` value for the item. For failures, the second-level status codes MAY simply be "*Failed*" or they may indicate, with more detail, the reason for the failure.

  If the top-level status is "*Failed*," second level status codes MAY be present which contain detailed error information if the ProvS wants to share that information with the invoking party.

4.12. Operation: *PMEDelete*

The *PMEDelete* operation is used by the PM-issuing authority to delete an existing PMEngine at the ProvS.

4.12.1. `wsa:Action` values for `PMEDelete` Messages

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

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

4.12.2. `PMEDelete` Message

The `<PMEDelete>` request is called to delete the specified PMEngine(s) at the ProvS.

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

- `<PMEngineRef>` **[Required]** - one or more of the reference(s) to the PMEngine(s) to be deleted.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMEDelete - retreive the specified PMEngine -->

<xs:element name="PMEDelete" type="PMEDeleteType"/>

<xs:complexType name="PMEDeleteType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"  maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 19. `<prov:PMEDelete>` Schema Fragment

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

```
<prov:PMEDelete>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.6</prov:PMEngineRef>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.5</prov:PMEngineRef>
</prov:PMEDelete>
```

Example 19. Example `<prov:PMEDelete>` Message

4.12.3. `PMEDeleteResponse` Message

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

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

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

```
<!-- PMEDeleteResponse - response for the PMEDelete Request -->

<xs:element name="PMEDeleteResponse" type="PMEDeleteResponseType"/>

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

Figure 20. `<prov:PMEDeleteResponse>` Schema Fragment

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

```
<prov:PMEDeleteResponse>
  <lu:Status code="OK" />
</prov:PMEDeleteResponse>
```

Example 20. Example `<prov:PMEDeleteResponse>` Message

4.12.4. PMEDelete Processing Rules

- The ProvS SHOULD take steps to ensure that the correct party (typically the party that registered and uploaded the PMEngine) is submitting the request and to reject requests from inappropriate parties. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Forbidden*."

- Requests which include a `<PMEngineRef>` that is not known to the ProvS, MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."

- If the engines pointed to by all `<PMEngineRef>` elements in the request are successfully deleted, the top-level status code MUST be "*OK*." If all of the deletes failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was NOT successful indicating so and including the `ref` attribute containing the `<prov:PMEngineRef>` value for the item. These second-level status codes MAY simply be "*Failed*," or they may indicate, with more detail, the reason for the failure.

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

4.13. Operation: *PMEDisable*

The *PMEDisable* operation is used by the PM-issuing authority to disable an existing PMEngine at the ProvS. When a PMEngine is disabled, it is not available for further downloads.

4.13.1. `wsa:Action` values for `PMEDisable` Messages

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

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

4.13.2. `PMEDisable` Message

The `<PMEDisable>` request is called to disable the specified PMEngine(s) at the ProvS.

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

- `<PMEngineRef>` **[Required]** - one or more of the reference(s) to the PMEngine(s) to be disabled.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMEDisable - retreive the specified PMEngine -->

<xs:element name="PMEDisable" type="PMEDisableType"/>

<xs:complexType name="PMEDisableType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 21. `<prov:PMEDisable>` Schema Fragment

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

```
<prov:PMEDisable>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.6</prov:PMEngineRef>
</prov:PMEDisable>
```

Example 21. Example `<prov:PMEDisable>` Message

4.13.3. `PMEDisableResponse` Message

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

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

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

```
<!-- PMEDisableResponse - response for the PMEDisable Request -->

<xs:element name="PMEDisableResponse" type="PMEDisableResponseType"/>

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

Figure 22. `<prov:PMEDisableResponse>` Schema Fragment

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

```
<prov:PMEDisableResponse>
  <lu:Status code="OK" />
</prov:PMEDisableResponse>
```

Example 22. Example `<prov:PMEDisableResponse>` Message

4.13.4. PMEDisable Processing Rules

- The ProvS SHOULD take steps to ensure that the correct party (typically the party that registered and uploaded the PMEngine) is submitting the request and to reject requests from inappropriate parties. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Forbidden*."

- The ProvS MUST NOT allow subsequent downloads of the specified PMEngine following the completion of this call and SHOULD abandon any in-progress downloads that have not yet completed.

- Requests to disable a PMEngine that is not known to the ProvS MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."

- Requests to disable a PMEngine that is not enabled MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Duplicate*."

- If the engines pointed to by **ALL** `<PMEngineRef>` elements in the request are successfully disabled, the top-level status code MUST be "*OK*." If all of the disable operations failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was NOT successful indicating so and including the `ref` attribute containing the `<prov:PMEngineRef>` value for the item. These second-level status codes MAY simply be "*Failed*," or they may indicate, with more detail, the reason for the failure.

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

4.14. Operation: *PMEDownload*

The *PMEDownload* operation is used by the PMM to obtain the executable code for a PMEngine that is not currently instantiated within the PMM's domain.

4.14.1. `wsa:Action` values for `PMEDownload` Messages

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

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

4.14.2. `PMEDownload` Message

The `<PMEDownload>` request is called to obtain the executable code for a PMEngine referenced by a `<PMEngineRef>`.

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

- `<PMEngineRef>` **[Required]** - the reference to the desired PMEngine. This is typically taken from the element with the same name in a PMD that is being provisioned or updated at the PMM.

- `dp:BasicPagingAttributeGroup` **[Optional]** - a set of attributes supporting the pagination of results. This adaptation of the pagination design pattern from the Liberty ID-WSF Design Patterns [LibertyDP] specification uses the Basic Pagination model. The objects being downloaded are already static and so there is no need for the static result set support.

  For the purpose of pagination, the "item" being paginated is each byte of the executable being downloaded. So a pagination request with a `count` of *100* is requesting at most 100 bytes of the executable file. This count refers to the number of bytes prior to the base64 encoding necessary to place the bytes into a message. (The encoded length for this data is typically 33% larger.)

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

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

```
<!-- PMEDownload - download the specified PMEngine (or part thereof) -->

<xs:element name="PMEDownload" type="PMEDownloadType"/>

<xs:complexType name="PMEDownloadType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"/>
      </xs:sequence>
      <xs:attributeGroup ref="dp:BasicPagingAttributeGroup"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 23. `<prov:PMEDownload>` Schema Fragment

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

```
<prov:PMEDownload count="102400">
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
</prov:PMEDownload>
```

Example 23. Example `<prov:PMEDownload>` Message

A second example message body containing a `<PMEDownload>` message follows. This request asks for the first remaining bytes of the executable.

```
<prov:PMEDownload count="83276" offset="102400">
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
</prov:PMEDownload>
```

Example 24. 2nd Example `<prov:PMEDownload>` Message

4.14.3. `PMEDownloadResponse` Message

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

- `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- `<prov:EngineData>` **[Optional]** - the Base64 encoded bytes of the PMEngine executable. This MAY contain only a portion of the complete PMEngine depending upon the settings for the pagination attributes on the request.
- `dp:BasicPagingResponseAttributeGroup` **[Optional]** - a set of response attributes supporting the pagination of results. This adaptation of the pagination design pattern from the Liberty ID-WSF Design Patterns [LibertyDP] specification uses the Basic Pagination model.

  For the purpose of pagination, the "item" being paginated is each byte of the executable being downloaded. So a response with the pagination attribute `remaining` set to *8192* indicates that there are 8,192 bytes of data left, after the bytes included in the current response, to complete the download of the executable file.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- PMEDownloadResponse - response for the PMEDownload Request -->

<xs:element name="PMEDownloadResponse" type="PMEDownloadResponseType"/>

<xs:complexType name="PMEDownloadResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="EngineData" minOccurs="0" />
      </xs:sequence>
      <xs:attributeGroup ref="dp:BasicPagingResponseAttributeGroup"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="EngineData" type="xs:base64Binary"/>
```

Figure 24. `<prov:PMEDownloadResponse>` Schema Fragment

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

```
<prov:PMEDownloadResponse remaining="83276" nextOffset="102400">
  <lu:Status code="OK"/>
  <prov:EngineData>
   ... base64 encoded data (100K worth) ...
  </prov:EngineData>
</prov:PMEDownloadResponse>
```

Example 25. Example `<prov:PMEDownloadResponse>` Message

A second example message body containing a `<PMEDownloadResponse>` message follows. This is a successful response which includes the remaining 83,276 bytes for the file and indicates that there are no further bytes.

```
<prov:PMEDownloadResponse remaining="0">
  <lu:Status code="OK"/>
  <prov:EngineData>
   ... base64 encoded data (83276 bytes worth) ...
  </prov:EngineData>
</prov:PMEDownloadResponse>
```

Example 26. 2nd Example `<prov:PMEDownloadResponse>` Message

4.14.4. PMEDownload Processing Rules

- All of the processing rules defined for pagination in the Liberty ID-WSF Design Patterns [LibertyDP] are incorporated by reference and must be met.
- Requests which include a `<PMEngineRef>` that is not known to the ProvS, or not enabled for download, MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."
- Requests which include a `<PMEngineRef>` that refers to a PMEngine which has no engine data available (i.e., the size was zero in registration indicating that this Engine is built-in or available to the PMM through some other means) MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NoDownload*."
- If a PMEngine becomes disabled following the start of a download, but prior to the completion of the download, the ProvS SHOULD treat any subsequent requests for the additional segments of the file as a failure. In such cases, if detailed status codes are to be included, the detailed error code for this error MUST be "*NotFound*."
- If request processing succeeded, the top-level status code MUST be "*OK*." Otherwise, the top-level status code MUST be "*Failed*"
- If the top-level status code is "*Failed*," the response MAY also contain *Forbidden*, *NoDownload*, or *NotFound* as a second-level status code. The Provisioning Service instance may not wish to reveal the reason for failure, in which case, no second-level status code will appear.

4.15. Operation: *PMEEnable*

The *PMEEnable* operation is used by the PM-issuing authority to enable downloading of a PMEngine at the ProvS. This operation MUST NOT take place until the PMEngine has been completely uploaded to the ProvS.

4.15.1. `wsa:Action` values for `PMEEnable` Messages

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

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

4.15.2. `PMEEnable` Message

The `<PMEEnable>` request is called to enable downloading of the specified PMEngine(s) at the ProvS.

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

- `<PMEngineRef>` **[Required]** - one or more of the reference(s) to the PMEngine(s) to be enabled.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMEEnable - retreive the specified PMEngine -->

<xs:element name="PMEEnable" type="PMEEnableType"/>

<xs:complexType name="PMEEnableType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"  maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 25. `<prov:PMEEnable>` Schema Fragment

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

```
<prov:PMEEnable>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.6</prov:PMEngineRef>
</prov:PMEEnable>
```

Example 27. Example `<prov:PMEEnable>` Message

4.15.3. `PMEEnableResponse` Message

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

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

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

```
<!-- PMEEnableResponse - response for the PMEEnable Request -->

<xs:element name="PMEEnableResponse" type="PMEEnableResponseType"/>

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

Figure 26. `<prov:PMEEnableResponse>` Schema Fragment

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

```
<prov:PMEEnableResponse>
  <lu:Status code="OK"/>
</prov:PMEEnableResponse>
```

Example 28. Example `<prov:PMEEnableResponse>` Message

4.15.4. PMEEnable Processing Rules

- The ProvS SHOULD take steps to ensure that the correct party (typically the party that registered and uploaded the PMEngine) is submitting the request and to reject requests from inappropriate parties. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Forbidden*."

- Attempts to enable a PMEngine that is already enabled MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Duplicate*."

- The ProvS MUST NOT allow a PMEngine to be enabled if the registered size is greater than zero and the engine data of the same size is not available. If this is attempted, the enable attempt MUST be treated as a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*WrongSize*."

  Note that how the engine data is available to the ProvS is not an issue here – the data may have been uploaded using the `<PMEUpload>` interface or the ProvS may have some other means of acquiring the data. However, the size MUST still match regardless of how the engine data is acquired.

- If the size specified on the registration of the PMEngine is non-zero, the ProvS MUST NOT allow a PMEngine to be enabled when the hash value of the available engine data (calculated using the hash method specified during registration) does not match the registered hash value. If the calculated and registered hash values do not match, the enable attempt MUST be treated as a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*HashMismatch*."

  The hash digest MUST be calculated on the binary (decoded) bytes of the executable, NOT the Base64 encoded bytes.

- The ProvS MUST NOT allow subsequent uploads targeted at the specified PMEngine(s) following the completion of this call.

- Attempts to enable a PMEngine that is not known to the ProvS, MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."

- If the engines pointed to by **ALL** of the <PMEngineRef> elements in the request are successfully enabled, the top-level status code MUST be "*OK*." If all of the enable operations failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was NOT successful indicating so and including the ref attribute containing the <prov:PMEngineRef> value for the item. These second-level status codes MAY simply be "*Failed*" or they may indicate, with more detail, the reason for the failure.

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

4.16. Operation: *PMEGetInfo*

The *PMEGetInfo* operation is used to obtain information about a registered PMEngine.

4.16.1. `wsa:Action` values for `PMEGetInfo` Messages

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

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

4.16.2. `PMEGetInfo` Message

The <PMEGetInfo> request is called to get information about the specified PMEngine(s) at the ProvS.

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

- <PMEngineRef> **[Optional]** - zero or more reference(s) to the PMEngine(s) for which the current information is desired. If this element is not specified, the request is a query for all PMEngine(s) registered by the invoker.

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

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

```
<!-- PMEGetInfo - to get information about the specifed PME(s) -->

<xs:element name="PMEGetInfo" type="PMEGetInfoType"/>
<xs:complexType name="PMEGetInfoType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 27. <prov:PMEGetInfo> Schema Fragment

An example message body containing a <PMEGetInfo> message follows.

```
<prov:PMEGetInfo>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
</prov:PMEGetInfo>
```

Example 29. Example `<prov:PMEGetInfo>` Message

4.16.3. `PMEGetInfoResponse` Message

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

- `<lu:Status>` **[Required]** - the status of the response. See the processing rules below for more information.
- `<prov:PMEInfo>` **[Optional]** - zero or more elements containing the information about each PMEngine (see Section 3.5).
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

```
<!-- PMEGetInfoResponse - response to the PMEGetInfo request -->

<xs:element name="PMEGetInfoResponse" type="PMEGetInfoResponseType"/>

<xs:complexType name="PMEGetInfoResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMEInfo" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 28. `<prov:PMEGetInfoResponse>` Schema Fragment

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

```
<prov:PMEGetInfoResponse>
  <lu:Status code="OK" />
  <prov:PMEInfo>
    <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
    <prov:PMECreatorID>http://reg.providers.com"</prov:PMECreatorID>
    <prov:PMEWhenCreated>2007-01-18T17:32:14Z</prov:PMEWhenCreated>
    <prov:PMEEnabled>true</prov:PMEEnabled>
    <prov:PMEWhenEnabled>2007-01-18T18:15:22Z</prov:PMEWhenEnabled>
    <prov:PMESize>185676</prov:PMESize>
    <prov:PMEHash method="...SHA-1">...SHA1 hash data ...</prov:PMEHash>
  </prov:PMEInfo>
</prov:PMEGetInfoResponse>
```

Example 30. Example `<prov:PMEGetInfoResponse>` Message

4.16.4. PMEGetInfo Processing Rules

- Requests which do not include at least one PMEngineRef element are interpreted as a request for all PMEngine(s) registered by the invoker.

- Attempts to obtain information about a PMEngine that is not known to the ProvS MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."

- If the engines pointed to by **all** of the `<PMEngineRef>` elements in the request are successfully enabled, the top-level status code MUST be "*OK*." If all of the enable operations failed, the top-level status code MUST be "*Failed*." Otherwise, if the results were mixed, the top-level status MUST be "*Partial*" and the second level status MUST be included for items for which the processing was NOT successful indicating so and including the `ref` attribute containing the `<prov:PMEngineRef>` value for the item. These second-level status codes MAY simply be "*Failed*," or they may indicate, with more detail, the reason for the failure.

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

4.17. Operation: *PMERegister*

The *PMERegister* operation is used by the PM-issuing authority to register a new PMEngine at the ProvS. The PMEngine must first be registered before it can be uploaded to the ProvS.

4.17.1. `wsa:Action` values for `PMERegister` Messages

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

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

4.17.2. `PMERegister` Message

The `<PMERegister>` request is called to register the specified PMEngine at the ProvS.

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

- `<PMEngineRef>` **[Required]** - the reference id assigned to the PMEngine to be registered.

- `<PMESize>` **[Required]** - the size, in bytes, of the PMEngine's executable. This is the number of bytes that will be uploaded via the `<prov:PMEUpload>` interface for this PMEngine. If a PMEngine is not to be uploaded, but is, in fact, built-in to the target platform(s), the size should be defined as zero.

- `<PMEHash>` **[Required]** - the digest hash of the PMEngine's executable. This is used to verify that the executable has been uploaded properly and/or by the PMM to ensure that its local image is correct. The hash methods supported are the same methods defined for XML Digital Signature, most notably SHA-1 and MD5.

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

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

```
<!-- PMERegister - retreive the specified PMEngine -->

<xs:element name="PMERegister" type="PMERegisterType"/>

<xs:complexType name="PMERegisterType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"/>
        <xs:element ref="PMESize"   />
        <xs:element ref="PMEHash"   />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 29. <prov:PMERegister> Schema Fragment

An example message body containing a <PMERegister> message follows.

```
<prov:PMERegister>
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.6</prov:PMEngineRef>
  <prov:PMESize>185676</prov:PMESize>
  <prov:PMEHash method="http://www.w3.org/2000/09/xmldsig#sha-1">
    ...SHA1 hash data ...
  </prov:PMEHash>
</prov:PMERegister>
```

Example 31. Example <prov:PMERegister> Message

4.17.3. PMERegisterResponse Message

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

- <lu:Status> **[Required]** - the status of the response. See the processing rules below for more information.
- <prov:PMEUploadMax> **[Optional]** - the upper limit on the number of bytes that the caller should send in a single <prov:PMEUpload> invocation. This element MUST be specified if the request was successful and it included a non-zero <prov:PMESize> value.
- anyAttribute **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **xs:ID** type attribute such as xml:id or wsu:Id.

```
<!-- PMERegisterResponse - response for the PMERegister Request -->

<xs:element name="PMERegisterResponse" type="PMERegisterResponseType"/>
<xs:complexType name="PMERegisterResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMEUploadMax" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMEUploadMax" type="xs:integer"/>
```

Figure 30. `<prov:PMERegisterResponse>` Schema Fragment

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

```
<prov:PMERegisterResponse>
  <lu:Status code="OK"/>
  <prov:PMEUploadMax>102400</prov:PMEUploadMax>
</prov:PMERegisterResponse>
```

Example 32. Example `<prov:PMERegisterResponse>` Message

4.17.4. PMERegister Processing Rules

- The ProvS SHOULD take steps to ensure that the correct party is submitting the request and to reject requests from inappropriate parties. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Forbidden*."

- Requests which include a `<PMEngineRef>` that is already known to the ProvS, MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*Duplicate*."

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

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

4.18. Operation: *PMEUpload*

The *PMEUpload* operation is used by the PM-issuing authority to upload the executable code for a PMEngine to the ProvS.

4.18.1. `wsa:Action` values for `PMEUpload` Messages

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

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

4.18.2. `PMEUpload` Message

The `<PMEUpload>` request is called to store the executable code module for a PMEngine at the ProvS so that it can be retrieved by the PMM using the `<PMEDownload>` interface.

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

- `<PMEngineRef>` **[Required]** - the identity of the PMEngine assigned by the issuer.
- `<prov:EngineData>` **[Required]** - the Base64 encoded bytes of the PMEngine executable. This MAY contain only a portion of the complete PMEngine if the file is larger than the number of bytes allowed for an upload.

  All counts of bytes and offsets are related to the unencoded bytes of the executable. The Base64 encoding will typically add approximately 33% additional bytes (e.g., a count of 300 bytes would result in an encoded string of approximately 400 bytes).

  The number of executable bytes (prior to encoding) SHOULD NOT exceed the limit specified in the `<prov:PMEUploadMax>` element in the `<prov:PMERegisterResponse>` message where this PMEngine was registered.
- `offset` **[Required]** - the zero based offset in the uploaded file for the first byte of data in the enclosed `<prov:EngineData>` element.
- `remaining` **[Required]** - the number of bytes remaining to be uploaded for the file after the enclosed data bytes are added to the file.
- `anyAttribute` **[Optional]** - zero or more attributes from a namespace other than that of this specification. One such possibility is an **`xs:ID`** type attribute such as `xml:id` or `wsu:Id`.

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

```
<!-- PMEUpload - retreive the specified PMEngine -->

<xs:element name="PMEUpload" type="PMEUploadType"/>

<xs:complexType name="PMEUploadType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"/>
        <xs:element ref="EngineData" />
      </xs:sequence>
      <xs:attribute name="offset"   use="required" type="xs:integer"/>
      <xs:attribute name="remaining" use="required" type="xs:integer"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 31. <prov:PMEUpload> Schema Fragment

An example message body containing a <PMEUpload> message follows. This request contains the first (offset 0) 64K bytes of a 185,676 byte file (65,536 in this message plus 120,140 remaining).

```
<prov:PMEUpload offset="0" remaining="120140">
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
  <prov:EngineData>
    ... 64K of base 64 encoded data
  </prov:EngineData>
</prov:PMEUpload>
```

Example 33. Example <prov:PMEUpload> Message

A second example message body containing a <PMEUpload> message follows. This request includes the next 64K of the same 185,676 byte executable.

```
<prov:PMEUpload offset="65536" remaining="54604">
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
  <prov:EngineData>
   ... base64 encoded data (64K worth) ...
  </prov:EngineData>
</prov:PMEUpload>
```

Example 34. Example <prov:PMEUpload> Message

A third example message body containing a <PMEUpload> message follows. This request includes the remaining bytes of the same 185,676 byte executable.

```
<prov:PMEUpload offset="131072" remaining="0">
  <prov:PMEngineRef>http://pmsRus.org/VeryTrustedModule/3.7</prov:PMEngineRef>
  <prov:EngineData>
   ... base64 encoded data (54604 bytes worth) ...
  </prov:EngineData>
</prov:PMEUpload>
```

Example 35. Example <prov:PMEUpload> Message

4.18.3. `PMEUploadResponse` Message

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

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

```
<!-- PMEUploadResponse - response for the PMEUpload Request -->

<xs:element name="PMEUploadResponse" type="PMEUploadResponseType"/>

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

Figure 32. `<prov:PMEUploadResponse>` Schema Fragment

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

```
<prov:PMEUploadResponse>
  <lu:Status code="OK"/>
</prov:PMEUploadResponse>
```

Example 36. Example `<prov:PMEUploadResponse>` Message

4.18.4. PMEUpload Processing Rules

- Attempts to upload data for a PMEngine that is not known to the ProvS MUST result in a failure. If detailed status codes are to be included in the response, the detailed error code for this error MUST be "*NotFound*."
- The invoking party SHOULD NOT exceed the limit placed on the maximum number of upload bytes of data set by the ProvS in the `<prov:PMEUploadMax>` element in the `<prov:PMERegisterResponse>` message when this PMEngine was registered. If the invoker does exceed the limit, the ProvS MAY accept the invocation as-is, MAY generate a SOAP fault, or MAY otherwise treat this as a failure. If this is treated as a failure and not a SOAP fault and the ProvS is including detailed error status codes, the detailed status code for this error MUST be "*LimitExceeded*."
- The invoking party MAY restart the uploading of a file if the PMEngine has not been enabled for download (perhaps because their upload process was reset) by sending an upload request with an offset of zero. When this occurs, the ProvS must throw away previously uploaded data and start anew with the data from the current packet.
- If the calculated (using the remaining count, offset and data length) or actual size of the uploaded data differs from the size that was registered for the PMEngine, the ProvS SHOULD treat the upload as a failure. If it is treated as a failure and detailed status codes are included in the results, the detailed status code for this error MUST be "*WrongSize*."

- Other than an upload restart as outlined above, the invoking party MUST send each segment of the uploaded file in sequential order, without skipping a section or otherwise transmitting the segments out-of-order. If a segment is transmitted out-of-order (where the first byte of the new segment is not adjacent to the last byte of the previous segment), the ProvS MUST treat this operation as a failure. If the ProvS is including detailed error status codes, the detailed status code for this error MUST be "*OutOfOrder*."

- Once all segments of an executable have been uploaded and the executable has been enabled, the ProvS SHOULD NOT allow the executable to be changed. If an attempt is made to upload new data for an already uploaded executable, the ProvS SHOULD treat such requests as a failure. If the ProvS is including detailed error status codes, the detailed status code for this error MUST be "*Duplicate*."

  Note that if the invoking party, for some reason, decides that they have uploaded the wrong file, they would have to disable the existing PMEngine and upload the correct file with a new, non-conflicting, `<prov:PMEngineRef>`. This is to prevent the possibility of ever having two different executable files with the exact same `<prov:PMEngineRef>`.

- The Provisioning Service SHOULD take steps to ensure that the correct party is submitting the request and to reject requests from inappropriate parties.

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

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

4.19. Operation: *Poll*

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

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

4.19.1. `wsa:Action` values for `Poll` Messages

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

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

4.19.2. `Poll` Message

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

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

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

```
<!-- Poll - Poll for new service requests -->

<xs:element name="Poll" type="dp:PollType"/>
```

Figure 33. `<prov:Poll>` — Schema Fragment

An example message body containing a <prov:Poll> message follows. This is a request asking for <pmm:PMUpdate>, <pmm:PMDelete>, or <pmm:PMGetStatus> requests and asks ProvS to wait for 5 minutes if none are immediately available.

```
<prov:Poll wait="300">
  <wsa:Action>urn:liberty:pmm:2007-09:PMUpdate</wsa:Action>
  <wsa:Action>urn:liberty:pmm:2007-09:PMDelete</wsa:Action>
  <wsa:Action>urn:liberty:pmm:2007-09:PMGetStatus</wsa:Action>
</prov:Poll>
```

Example 37. Example <prov:Poll> Message

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

```
<prov:Poll wait="300">
  <wsa:Action>urn:liberty:pmm:2007-09:PMUpdate</wsa:Action>
  <wsa:Action>urn:liberty:pmm:2007-09:PMDelete</wsa:Action>
  <wsa:Action>urn:liberty:pmm:2007-09:PMGetStatus</wsa:Action>
  <dp:Response ref="1">
    <prov:UpdatePMResponse>
      <lu:Status code="WillNotify" />
    </prov:UpdatePMResponse>
  </dp:Response>
</prov:Poll>
```

Example 38. Example <prov:Poll> Message with a <prov:UpdatePMResponse>.

4.19.3. PollResponse Message

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

```
  <!--PollResponse - response for the Poll request -->

  <xs:element name="PollResponse" type="dp:PollResponseType"/>
```

Figure 34. <prov:PollResponse> — Schema Fragment

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

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

Example 39. Example <prov:PollResponse> Message

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

```
<prov:PollResponse>
  <lu:Status code="OK" />
  <dp:Request itemID="1">
    <pmm:PMUpdate>
      <pmm:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
        <prov:PMDescriptor xs:id="2323923900239" >
          <prov:PMID issuer="http://provs-r-us.com">uuid:778349-283920-88379-5448739</prov:PMID>
          <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
          <ds:Signature>
            ... signature data goes here ...
          </ds:Signature>
        </prov:PMDescriptor>
      </pmm:PMUpdateItem>
      <dp:NotifyTo>
        <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      </dp:NotifyTo>

    </pmm:PMUpdate>
  </dp:Request>
</prov:PollResponse>
```

Example 40. Example `<prov:PollResponse>` Message

4.19.4. Poll Processing Rules

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

4.20. Operation: *UpdateEPR*

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

4.20.1. `wsa:Action` values for `UpdateEPR` Messages

`<UpdateEPR>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:prov:2007-09:UpdateEP`

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

4.20.2. `UpdateEPR` Message

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

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

- `<PMID>` **[Required]** - the identifier of the PM to which this update is related. This identifier is included in the `<prov:PMDescriptor>` returned in the `<prov:PMGetDescriptorResponse>`.

- `<prov:CallbackEPR>` **[Required]** - one or more updated ID-WSF EPR(s). This is a complete replacement of any formerly registered EPRs related to this PM.

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

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

```
<!-- UpdateEPR - update the EPR for PM maintenance operations -->

<xs:element name="UpdateEPR" type="UpdateEPRType"/>

<xs:complexType name="UpdateEPRType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="UpdateEPRItem" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="UpdateEPRItem" type="UpdateEPRItemType" />

<xs:complexType name="UpdateEPRItemType">
  <xs:sequence>
    <xs:element ref="PMID"      />
    <xs:element ref="CallbackEPR"  />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>
```

Figure 35. <prov:UpdateEPR> — Schema Fragment

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

```
<prov:UpdateEPR>
  <prov:UpdateEPRItem itemID="1" >
    <prov:PMID issuer="http://provs-r-us.com">uuid:778349-28392 0-88379-5448739</prov:PMID>
    <prov:CallbackEPR>
      <wsa:Address>http://im-a-pmm. com</wsa:Address>
      <wsa:Metadata>
        <ds:ServiceType>urn:liberty:pmm:2007- 09</ds:ServiceType>
        <ds:Framework version="2.0" />
        <ds:SecurityContext>
          <ds:SecurityMechID>urn:liberty:security:2006 -08:TLS:SAMLV2</ds:Security MechID>
        </ds:SecurityContext>
     </wsa:Metadata>
    </prov:CallbackEPR>
  </prov:UpdateEPRItem>
  <prov:UpdateEPRItem itemID="2">
    <prov:PMID issuer="http://provs-r-us.com">uuid:239032-230 328-92379-2397923</prov:PMID>
    <prov:CallbackEPR>
      <wsa:Address>http://im-a-pmm.com</wsa:Address>
      <wsa:Metadata>
        <ds:ServiceType>urn:liberty:pmm:200 7-09</ds:ServiceType>
        <ds:Framework version="2.0" />
        <ds:SecurityContext>
          <ds:SecurityMechID>urn:liberty:security:20 06-08:TLS:SAMLV2</ds:Securi tyMechID>
        </ds:SecurityContext>
     </wsa:Metadata>
    </prov:CallbackEPR>
  </prov:UpdateEPRItem>
</prov:UpdateEPR>
```

Example 41. Example <prov:UpdateEPR> Message

4.20.3. `UpdateEPRResponse` Message

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

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

```
<!-- UpdateEPRResponse - the response to the UpdateEPR request -->

<xs:element name="UpdateEPRResponse" type="UpdateEPRResponseType"/>

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

Figure 36. `<prov:UpdateEPRResponse>` — Schema Fragment

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

```
<prov:UpdateEPRResponse>
  <lu:Status code="Partial">
    <lu:Status code="OK"       ref="1" />
    <lu:Status code="NotFound" ref="2" />
  </lu:Status>
</prov:UpdateEPRResponse>
```

Example 42. Example `<prov:UpdateEPRResponse>` Message

4.20.4. UpdateEPR Processing Rules

- If, for any reason, an update is not accepted, the existing service instance at the ProvS MUST NOT be affected.
- PMs instantiated in one PMM MUST NOT be visible to other PMMs. If a PMM presents a PMID for a PM instance that has been instantiated in a different PMM, the ProvS MUST behave as if that PM does not exist and, accordingly, any attempt to update the associated CallbackEPR should fail. In such cases, the request MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this case MUST be "*NotFound*."
- The ProvS SHOULD generate an error if the CallbackEPR is such that the ProvS is unable to meet the requirements within the ID-WSF EPR and, therefore, the ProvS would be unable to dereference the ID-WSF EPR (for example, if the ProvS does not support the framework version(s) specified in the CallbackEPR). If detailed status codes are included in the response, the detailed status code for this case MUST be "*FeatureNotSupported*."
- Each `<prov:UpdateEPRItem>` is processed independently and may, independently, succeed or fail on its own merits.

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

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

5. Provisioning Service Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:prov:2007-09"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:prov="urn:liberty:prov:2007-09"
   xmlns:dp="urn:liberty:dp:2007-09"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns="urn:liberty:prov:2007-09"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>

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

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

  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
     schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.
xsd"/>

  <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
    schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
  <xs:import namespace="http://www.w3.org/2005/08/addressing"
    schemaLocation="http://www.w3.org/2005/08/addressing/ws-addr.xsd" />

  <!-- PMID - the Provisioned Module IDentifier  -->

  <xs:element name="PMID" type="PMIDType" />
  <xs:complexType name="PMIDType">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:attribute name="issuer" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


  <!-- PMDescriptor - describes/carries the components of a PM -->

  <xs:element name="PMDescriptor" type="PMDescriptorType"/>

  <xs:complexType name="PMDescriptorType">
    <xs:sequence>
      <xs:element ref="ds:Signature" minOccurs="0"/>
      <xs:element ref="PMID"                />
      <xs:element ref="PMEngineRef"  minOccurs="0"/>
      <xs:element ref="PMEngine"    minOccurs="0"/>
      <xs:element ref="PMInitData"   minOccurs="0"/>
      <xs:element ref="PMRTData"    minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="activate"    type="xs:boolean"  use="optional"/>
    <xs:attribute name="activateAt"   type="xs:dateTime" use="optional"/>
    <xs:attribute name="deactivateAt" type="xs:dateTime" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:element name="PMInitData"   type="PMDataType" />
  <xs:element name="PMRTData"    type="PMDataType" />

  <xs:complexType name="PMDataType" mixed="false">
```

```
  <xs:sequence>
    <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="PMEngine"    type="PMEngineType" />

<xs:complexType name="PMEngineType" mixed="false">
  <xs:sequence>
    <xs:element ref="PMEInfo"  />
    <xs:element ref="PMEBytes" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="PMEBytes"    type="xs:base64Binary" />


<!-- ProvisioningHandle - Info for PMM to initiate provisioning process -->

<xs:element name="ProvisioningHandle" type="ProvisioningHandleType"/>
<xs:complexType name="ProvisioningHandleType">
  <xs:sequence>
    <xs:element ref="PMDArtifact"  />
    <xs:element ref="ProvisioningServiceEPR" minOccurs="0"
                                maxOccurs="unbounded" />
    <xs:element ref="ds:Signature"        minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="expires" use="optional" type="xs:dateTime"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

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

<xs:element name="PMDArtifact" type="xs:string" />


<!-- CallbackEPR - where the PMM can receive Provisionig update requests -->

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


<!-- PMStatus - Provisioning status of the PM -->

<xs:element name="PMStatus" type="PMStatusType"/>
<xs:complexType name="PMStatusType">
  <xs:sequence>
    <xs:element ref="PMID"/>
    <xs:element ref="State"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="State" type="StateType"/>
<xs:complexType name="StateType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:attribute name="asof"  type="xs:dateTime" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>


<!-- PMEInfo - The current state of a PMEngine -->

<xs:element name="PMEInfo" type="PMEInfoType"/>
<xs:complexType name="PMEInfoType">
```

```
    <xs:sequence>
      <xs:element ref="PMEngineRef"/>
      <xs:element ref="PMECreatorID"/>
      <xs:element ref="PMEWhenCreated"/>
      <xs:element ref="PMEEnabled"/>
      <xs:element ref="PMEWhenEnabled"/>
      <xs:element ref="PMESize"/>
      <xs:element ref="PMEHash"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax" />
  </xs:complexType>

  <xs:element name="PMEngineRef" type="xs:anyURI"/>

  <xs:element name="PMECreatorID" type="xs:anyURI"/>

  <xs:element name="PMEWhenCreated" type="xs:dateTime"/>

  <xs:element name="PMEEnabled" type="xs:boolean"/>

  <xs:element name="PMEWhenEnabled" type="xs:dateTime"/>

  <xs:element name="PMESize" type="xs:integer"/>

  <xs:element name="PMEHash" type="PMEHashType"/>
  <xs:complexType name="PMEHashType">
    <xs:simpleContent>
      <xs:extension base="xs:base64Binary">
        <xs:attribute name="method" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>


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

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


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

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


  <!--*********************************************************-->
  <!--                                                       -->
  <!--       PMEngine Maintenance interfaces                 -->
  <!--                                                       -->
  <!-- These interfaces are used to manage PMEngine objects at  -->
  <!-- the ProvS and include register/delete,enable/disable,    -->
  <!-- upload/download, and getInfo operations                  -->
  <!--                                                       -->
  <!--*********************************************************-->
  <!-- PMERegister - retreive the specified PMEngine -->

  <xs:element name="PMERegister" type="PMERegisterType"/>

  <xs:complexType name="PMERegisterType">
    <xs:complexContent>
```

```
    <xs:extension base="RequestAbstractType">
     <xs:sequence>
      <xs:element ref="PMEngineRef"/>
      <xs:element ref="PMESize"   />
      <xs:element ref="PMEHash"   />
     </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMERegisterResponse - response for the PMERegister Request -->

<xs:element name="PMERegisterResponse" type="PMERegisterResponseType"/>
<xs:complexType name="PMERegisterResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
     <xs:sequence>
      <xs:element ref="PMEUploadMax" minOccurs="0"/>
     </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMEUploadMax" type="xs:integer"/>


<!-- PMEUpload - retreive the specified PMEngine -->

<xs:element name="PMEUpload" type="PMEUploadType"/>

<xs:complexType name="PMEUploadType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
     <xs:sequence>
      <xs:element ref="PMEngineRef"/>
      <xs:element ref="EngineData" />
     </xs:sequence>
     <xs:attribute name="offset"   use="required" type="xs:integer"/>
     <xs:attribute name="remaining" use="required" type="xs:integer"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMEUploadResponse - response for the PMEUpload Request -->

<xs:element name="PMEUploadResponse" type="PMEUploadResponseType"/>

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


<!-- PMEDownload - download the specified PMEngine (or part thereof) -->

<xs:element name="PMEDownload" type="PMEDownloadType"/>

<xs:complexType name="PMEDownloadType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
     <xs:sequence>
      <xs:element ref="PMEngineRef"/>
     </xs:sequence>
     <xs:attributeGroup ref="dp:BasicPagingAttributeGroup"/>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMEDownloadResponse - response for the PMEDownload Request -->

<xs:element name="PMEDownloadResponse" type="PMEDownloadResponseType"/>

<xs:complexType name="PMEDownloadResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="EngineData" minOccurs="0" />
      </xs:sequence>
      <xs:attributeGroup ref="dp:BasicPagingResponseAttributeGroup"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="EngineData" type="xs:base64Binary"/>


<!-- PMEEnable - retreive the specified PMEngine -->

<xs:element name="PMEEnable" type="PMEEnableType"/>

<xs:complexType name="PMEEnableType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"  maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMEEnableResponse - response for the PMEEnable Request -->

<xs:element name="PMEEnableResponse" type="PMEEnableResponseType"/>

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


<!-- PMEDisable - retreive the specified PMEngine -->

<xs:element name="PMEDisable" type="PMEDisableType"/>

<xs:complexType name="PMEDisableType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMEDisableResponse - response for the PMEDisable Request -->

<xs:element name="PMEDisableResponse" type="PMEDisableResponseType"/>
```

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


<!-- PMEDelete - retreive the specified PMEngine -->

<xs:element name="PMEDelete" type="PMEDeleteType"/>

<xs:complexType name="PMEDeleteType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef"  maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMEDeleteResponse - response for the PMEDelete Request -->

<xs:element name="PMEDeleteResponse" type="PMEDeleteResponseType"/>

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


<!-- PMEGetInfo - to get information about the specifed PME(s) -->

<xs:element name="PMEGetInfo" type="PMEGetInfoType"/>
<xs:complexType name="PMEGetInfoType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMEngineRef" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMEGetInfoResponse - response to the PMEGetInfo request -->

<xs:element name="PMEGetInfoResponse" type="PMEGetInfoResponseType"/>

<xs:complexType name="PMEGetInfoResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMEInfo" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!--*************************************************************-->
<!--                                                             -->
<!--       Provisioned Module Maintenance interfaces             -->
<!--                                                             -->
```

```
<!-- These interfaces are used to manage PMs during and after  -->
<!-- the initial provisioning process.  Actions taken with    -->
<!-- these interfaces typically result in indirect actions    -->
<!-- within a PMM where the PM has been provisioned.        -->
<!--                                                        -->
<!--********************************************************-->
<!-- PMGetDescriptor - request to exchange PMHandle for PMGetDescriptor -->

<xs:element name="PMGetDescriptor" type="PMGetDescriptorType"/>

<xs:complexType name="PMGetDescriptorType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMDArtifact"/>
        <xs:element ref="CallbackEPR" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMGetDescriptorResponse - response to the PMGetDescriptor request -->

<xs:element name="PMGetDescriptorResponse" type="PMGetDescriptorResponseType"/>

<xs:complexType name="PMGetDescriptorResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMDescriptor" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMActivate - to activate one or more PM(s) at the PMM -->

<xs:element name="PMActivate" type="PMActivateType"/>

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

<xs:element name="PMActivateItem" type="PMActivateItemType" />

<xs:complexType name="PMActivateItemType">
  <xs:sequence>
    <xs:element ref="prov:PMID"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string"   use="required"/>
  <xs:attribute name="at"    type="xs:dateTime"  use="optional"/>
</xs:complexType>


<!-- PMActivateResponse - the response to the PMActivate request -->

<xs:element name="PMActivateResponse" type="PMActivateResponseType"/>
```

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


<!-- PMDeactivate - to deactivate a PM at the PMM -->

<xs:element name="PMDeactivate" type="PMDeactivateType"/>

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

<xs:element name="PMDeactivateItem" type="PMDeactivateItemType" />

<xs:complexType name="PMDeactivateItemType">
  <xs:sequence>
    <xs:element ref="prov:PMID"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string"   use="required"/>
  <xs:attribute name="at"    type="xs:dateTime"  use="optional"/>
</xs:complexType>


<!-- PMDeactivateResponse - the response to the PMDeactivate request -->

<xs:element name="PMDeactivateResponse" type="PMDeactivateResponseType"/>

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


<!-- PMRegisterDescriptor - to register a new PMD at the ProvS -->

<xs:element name="PMRegisterDescriptor" type="PMRegisterDescriptorType"/>

<xs:complexType name="PMRegisterDescriptorType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMRegisterDescriptorItem"  maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMRegisterDescriptorItem" type="PMRegisterDescriptorItemType" />

<xs:complexType name="PMRegisterDescriptorItemType">
  <xs:sequence>
    <xs:element ref="PMDescriptor" />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
</xs:complexType>
```

```
<!-- PMRegisterDescriptorResponse - response to the PMRegisterDescriptor request -->

<xs:element name="PMRegisterDescriptorResponse" type="PMRegisterDescriptorResponseType"/>

<xs:complexType name="PMRegisterDescriptorResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMRegisterDescriptorResponseItem" minOccurs="0"
                                      maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMRegisterDescriptorResponseItem"
         type="PMRegisterDescriptorResponseItemType"/>

<xs:complexType name="PMRegisterDescriptorResponseItemType">
  <xs:sequence>
    <xs:element ref="ProvisioningHandle"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:string" use="required"/>
</xs:complexType>


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

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

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

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

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


<!-- PMUpdateResponse - response to the PMUpdate request -->

<xs:element name="PMUpdateResponse" type="PMUpdateResponseType"/>

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


<!-- PMDelete - to register a new PM at the ProvS -->
```

```
<xs:element name="PMDelete" type="PMDeleteType"/>
<xs:complexType name="PMDeleteType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMDeleteItem"  maxOccurs="unbounded" />
        <xs:element ref="dp:NotifyTo"   minOccurs="0"      />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PMDeleteItem" type="PMDeleteItemType" />
<xs:complexType name="PMDeleteItemType">
  <xs:sequence>
    <xs:element ref="prov:PMID"/>
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string"   use="required"/>
</xs:complexType>


<!-- PMDeleteResponse - the response to the PMDelete request -->

<xs:element name="PMDeleteResponse" type="PMDeleteResponseType"/>

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


<!-- PMStatus - to get the status of a PM at the ProvS -->

<xs:element name="PMGetStatus" type="PMGetStatusType"/>

<xs:complexType name="PMGetStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="PMID"  minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMGetStatusResponse - response to the PMGetStatus request -->

<xs:element name="PMGetStatusResponse" type="PMGetStatusResponseType"/>

<xs:complexType name="PMGetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="PMStatus" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- PMSetStatus - update provisioning status of PM -->

<xs:element name="PMSetStatus" type="PMSetStatusType"/>

<xs:complexType name="PMSetStatusType">
```

```
  <xs:complexContent>
   <xs:extension base="RequestAbstractType">
    <xs:sequence>
     <xs:element ref="PMStatus"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>


 <!-- PMSetStatusResonse - response for PMSetStatus request -->

 <xs:element name="PMSetStatusResponse" type="PMSetStatusResponseType"/>

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


 <!-- Poll - Poll for new service requests -->

 <xs:element name="Poll" type="dp:PollType"/>


 <!--PollResponse - response for the Poll request -->

 <xs:element name="PollResponse" type="dp:PollResponseType"/>


 <!-- UpdateEPR - update the EPR for PM maintenance operations -->

 <xs:element name="UpdateEPR" type="UpdateEPRType"/>

 <xs:complexType name="UpdateEPRType">
  <xs:complexContent>
   <xs:extension base="RequestAbstractType">
    <xs:sequence>
     <xs:element ref="UpdateEPRItem" maxOccurs="unbounded" />
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:element name="UpdateEPRItem" type="UpdateEPRItemType" />

 <xs:complexType name="UpdateEPRItemType">
  <xs:sequence>
   <xs:element ref="PMID"       />
   <xs:element ref="CallbackEPR"  />
  </xs:sequence>
  <xs:attribute name="itemID" type="xs:string" use="required" />
 </xs:complexType>


 <!-- UpdateEPRResponse - the response to the UpdateEPR request -->

 <xs:element name="UpdateEPRResponse" type="UpdateEPRResponseType"/>

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

```
</xs:schema>
```

6. Provisioning Service WSDL

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

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


 <message name="PMGetDescriptor">
  <part name="body" element="prov:PMGetDescriptor"/>
 </message>
 <message name="PMGetDescriptorResponse">
  <part name="body" element="prov:PMGetDescriptorResponse"/>
 </message>

 <message name="PMActivate">
  <part name="body" element="prov:PMActivate"/>
 </message>
 <message name="PMActivateResponse">
  <part name="body" element="prov:PMActivateResponse"/>
 </message>

 <message name="PMDeactivate">
  <part name="body" element="prov:PMDeactivate"/>
 </message>
 <message name="PMDeactivateResponse">
  <part name="body" element="prov:PMDeactivateResponse"/>
 </message>

 <message name="PMDelete">
  <part name="body" element="prov:PMDelete"/>
 </message>
 <message name="PMDeleteResponse">
  <part name="body" element="prov:PMDeleteResponse"/>
 </message>

 <message name="PMGetStatus">
  <part name="body" element="prov:PMGetStatus"/>
 </message>
 <message name="PMGetStatusResponse">
  <part name="body" element="prov:PMGetStatusResponse"/>
 </message>

 <message name="PMRegisterDescriptor">
  <part name="body" element="prov:PMRegisterDescriptor"/>
 </message>
 <message name="PMRegisterDescriptorResponse">
  <part name="body" element="prov:PMRegisterDescriptorResponse"/>
 </message>
```

```
<message name="PMSetStatus">
  <part name="body" element="prov:PMSetStatus"/>
</message>
<message name="PMSetStatusResponse">
  <part name="body" element="prov:PMSetStatusResponse"/>
</message>

<message name="PMUpdate">
  <part name="body" element="prov:PMUpdate"/>
</message>
<message name="PMUpdateResponse">
  <part name="body" element="prov:PMUpdateResponse"/>
</message>

<message name="PMEDelete">
  <part name="body" element="prov:PMEDelete"/>
</message>
<message name="PMEDeleteResponse">
  <part name="body" element="prov:PMEDeleteResponse"/>
</message>

<message name="PMEDisable">
  <part name="body" element="prov:PMEDisable"/>
</message>
<message name="PMEDisableResponse">
  <part name="body" element="prov:PMEDisableResponse"/>
</message>

<message name="PMEDownload">
  <part name="body" element="prov:PMEDownload"/>
</message>
<message name="PMEDownloadResponse">
  <part name="body" element="prov:PMEDownloadResponse"/>
</message>

<message name="PMEEnable">
  <part name="body" element="prov:PMEEnable"/>
</message>
<message name="PMEEnableResponse">
  <part name="body" element="prov:PMEEnableResponse"/>
</message>

<message name="PMEGetInfo">
  <part name="body" element="prov:PMEGetInfo"/>
</message>
<message name="PMEGetInfoResponse">
  <part name="body" element="prov:PMEGetInfoResponse"/>
</message>

<message name="PMERegister">
  <part name="body" element="prov:PMERegister"/>
</message>
<message name="PMERegisterResponse">
  <part name="body" element="prov:PMERegisterResponse"/>
</message>

<message name="PMEUpload">
  <part name="body" element="prov:PMEUpload"/>
</message>
<message name="PMEUploadResponse">
  <part name="body" element="prov:PMEUploadResponse"/>
</message>

<message name="Poll">
  <part name="body" element="prov:Poll"/>
</message>
<message name="PollResponse">
```

```
  <part name="body" element="prov:PollResponse"/>
 </message>

 <message name="UpdateEPR">
  <part name="body" element="prov:UpdateEPR"/>
 </message>
 <message name="UpdateEPRResponse">
  <part name="body" element="prov:UpdateEPRResponse"/>
 </message>

 <portType name="ProvisioningPort">

  <operation name="PMGetDescriptor">
   <input message="tns:PMGetDescriptor"
     wsaw:Action="urn:liberty:prov:2007-09:PMGetDescriptor" />
   <output message="tns:PMGetDescriptorResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMGetDescriptorResponse" />
  </operation>

  <operation name="PMActivate">
   <input message="tns:PMActivate"
     wsaw:Action="urn:liberty:prov:2007-09:PMActivate" />
   <output message="tns:PMActivateResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMActivateResponse" />
  </operation>

  <operation name="PMDeactivate">
   <input message="tns:PMDeactivate"
     wsaw:Action="urn:liberty:prov:2007-09:PMDeactivate" />
   <output message="tns:PMDeactivateResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMDeactivateResponse" />
  </operation>

  <operation name="PMDelete">
   <input message="tns:PMDelete"
     wsaw:Action="urn:liberty:prov:2007-09:PMDelete" />
   <output message="tns:PMDeleteResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMDeleteResponse" />
  </operation>

  <operation name="PMGetStatus">
   <input message="tns:PMGetStatus"
     wsaw:Action="urn:liberty:prov:2007-09:PMGetStatus" />
   <output message="tns:PMGetStatusResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMGetStatusResponse" />
  </operation>

  <operation name="PMRegisterDescriptor">
   <input message="tns:PMRegisterDescriptor"
     wsaw:Action="urn:liberty:prov:2007-09:PMRegisterDescriptor" />
   <output message="tns:PMRegisterDescriptorResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMRegisterDescriptorResponse" />
  </operation>

  <operation name="PMSetStatus">
   <input message="tns:PMSetStatus"
     wsaw:Action="urn:liberty:prov:2007-09:PMSetStatus" />
   <output message="tns:PMSetStatusResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMSetStatusResponse" />
  </operation>

  <operation name="PMUpdate">
   <input message="tns:PMUpdate"
     wsaw:Action="urn:liberty:prov:2007-09:PMUpdate" />
   <output message="tns:PMUpdateResponse"
    wsaw:Action="urn:liberty:prov:2007-09:PMUpdateResponse" />
  </operation>
```

```
    <operation name="PMEDelete">
      <input message="tns:PMEDelete"
        wsaw:Action="urn:liberty:prov:2007-09:PMEDelete" />
      <output message="tns:PMEDeleteResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMEDeleteResponse" />
    </operation>

    <operation name="PMEDisable">
      <input message="tns:PMEDisable"
        wsaw:Action="urn:liberty:prov:2007-09:PMEDisable" />
      <output message="tns:PMEDisableResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMEDisableResponse" />
    </operation>

    <operation name="PMEDownload">
      <input message="tns:PMEDownload"
        wsaw:Action="urn:liberty:prov:2007-09:PMEDownload" />
      <output message="tns:PMEDownloadResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMEDownloadResponse" />
    </operation>

    <operation name="PMEEnable">
      <input message="tns:PMEEnable"
        wsaw:Action="urn:liberty:prov:2007-09:PMEEnable" />
      <output message="tns:PMEEnableResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMEEnableResponse" />
    </operation>

    <operation name="PMEGetInfo">
      <input message="tns:PMEGetInfo"
        wsaw:Action="urn:liberty:prov:2007-09:PMEGetInfo" />
      <output message="tns:PMEGetInfoResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMEGetInfoResponse" />
    </operation>

    <operation name="PMERegister">
      <input message="tns:PMERegister"
        wsaw:Action="urn:liberty:prov:2007-09:PMERegister" />
      <output message="tns:PMERegisterResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMERegisterResponse" />
    </operation>

    <operation name="PMEUpload">
      <input message="tns:PMEUpload"
        wsaw:Action="urn:liberty:prov:2007-09:PMEUpload" />
      <output message="tns:PMEUploadResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PMEUploadResponse" />
    </operation>

    <operation name="Poll">
      <input message="tns:Poll"
        wsaw:Action="urn:liberty:prov:2007-09:Poll" />
      <output message="tns:PollResponse"
        wsaw:Action="urn:liberty:prov:2007-09:PollResponse" />
    </operation>

    <operation name="UpdateEPR">
      <input message="tns:UpdateEPR"
        wsaw:Action="urn:liberty:prov:2007-09:UpdateEPR" />
      <output message="tns:UpdateEPRResponse"
        wsaw:Action="urn:liberty:prov:2007-09:UpdateEPRResponse" />
    </operation>

  </portType>

  <!--
```

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

<binding name="ProvisioningBinding" type="tns:ProvisioningPort">

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    </operation>

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

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

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

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

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

  </binding>

  <service name="ProvisioningService">

    <port name="ProvisioningPort" binding="tns:ProvisioningBinding">

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

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

    </port>

  </service>

</definitions>
```

References

Normative

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler, Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf*

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

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

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

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

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

Informative

[LibertyACT] Cahill, Conor P., eds. "Liberty Advanced Client Technologies," Version 1.0, Liberty Alliance Project (14 December 2007). *http://www.projectliberty.org/specs*

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

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

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