Liberty ID-WSF Provisioned Module Manager Service Specification

Version: 1.0

Editors:
Conor P. Cahill, Intel Corporation

Contributors:
George Fletcher, AOL LLC
Hubert Le Van Gong, Sun
Ben Lin, Intel Corporation
Paul Madsen, NTT
Hiroyoshi Takiguchi, NTT
Greg Whitehead, Hewlett-Packard
Janice Yang, Intel Corporation

Abstract:
This specification defines the interfaces for the Provisioning Module Manager (PMM).

Filename: liberty-idwsf-pmm-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.

Copyright © 2007 Adobe Systems; Agencia Catalana De Certificacio; America Online, Inc.; American Express
Company; Amsoft Systems Pvt Ltd.; Avatier Corporation; BIPAC; BMC Software, Inc.; Bank of America
Corporation; Beta Systems Software AG; British Telecommunications plc; Computer Associates International, Inc.;
Credentica; Dan Combs; Danish National IT and Telecom Agency; DataPower Technology, Inc.; Deutsche Telekom
AG, T-Com; Diamelle Technologies, Inc.; Diversinet Corp.; Drummond Group Inc.; Enosis Group LLC; Entrust,
Inc.; Epok, Inc.; Ericsson; Falkin Systems LLC; Fidelity Investments; Forum Systems, Inc.; France Télécom; French
Government Agency pour le développement de l’administration électronique (ADAE); Fugen Solutions, Inc; Fulvens
Ltd.; GSA Office of Governmentwide Policy; Gamefederation; Gemalto; General Motors; GeoFederation; Giesecke
& Devrient GmbH; Guy Huntington; Hewlett-Packard Company; Hochhauser & Co., LLC; IBM Corporation; Intel
Corporation; Intuit Inc.; Kantega; Kayak Interactive; Livo Technologies; Luminance Consulting Services; Mark
Wahl; Mary Ruddy; MasterCard International; MedCommons Inc.; Mobile Telephone Networks (Pty) Ltd; Mortgage
Bankers Association (MBA); NEC Corporation; NTT DoCoMo, Inc.; Netegrity, Inc.; NHK (Japan Broadcasting
Corporation) Science & Technical Research Laboratories; Neustar, Inc.; New Zealand Government State Services
Commission; Nippon Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; OpenNetwork; Oracle
Corporation; Ping Identity Corporation; Postsecondary Electronic Standards Council (PESC); RSA Security Inc.;
Reach; Reactivity Inc.; Rob Marano; Royal Mail Group plc; SAP AG; SanDisk Corporation; Senforce; Sharp
Laboratories of America; Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial
Trusted Network Technologies; UNINETT AS; UTI; VeriSign, Inc.; Vodafone Group Plc.; Wave Systems Corp.
Wells Fargo; All rights reserved.

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

1. Introduction ................................................................. 4
  1.1. Notation and Conventions .......................................... 4
    1.1.1. XML Namespaces ............................................ 4
2. Overview ................................................................. 5
  2.1. Provisioning Components ........................................ 5
3. Provisioned Module Manager Service (PMM) .......................... 7
  3.1. Service URIs ....................................................... 7
  3.2. Status Codes ...................................................... 7
  3.3. Data Definitions ................................................... 8
  3.4. Request and Response Abstract Types ......................... 8
    3.4.1. Complex Type RequestAbstractType .......................... 8
    3.4.2. Complex Type ResponseAbstractType ....................... 8
3.5. Operation: Provision ............................................ 9
  3.5.1. ws:Action values for Provision Messages .................. 9
  3.5.2. Provision Message ........................................... 9
  3.5.3. ProvisionResponse Message ................................ 10
  3.5.4. Provision Processing Rules .................................. 11
3.6. Operation: PMActivate ........................................... 12
  3.6.1. ws:Action values for PMActivate Messages ................. 12
  3.6.2. PMActivate Message ......................................... 12
  3.6.3. PMActivateResponse Message ................................. 13
  3.6.4. PMActivate Processing Rules ................................. 14
3.7. Operation: PMDeactivate ......................................... 15
  3.7.1. ws:Action values for PMDeactivate Messages ............... 15
  3.7.2. PMDeactivate Message ....................................... 15
  3.7.3. PMDeactivateResponse Message ............................... 16
  3.7.4. PMDeactivate Processing Rules ............................... 17
3.8. Operation: PMDelete .............................................. 18
  3.8.1. ws:Action values for PMDelete Messages .................... 18
  3.8.2. PMDelete Message ........................................... 18
  3.8.3. PMDeleteResponse Message .................................. 19
  3.8.4. PMDelete Processing Rules .................................. 20
3.9. Operation: PMUpdate .............................................. 20
  3.9.1. ws:Action values for PMUpdate Messages .................... 20
  3.9.2. PMUpdate Message ........................................... 20
  3.9.3. PMUpdateResponse Message .................................. 22
  3.9.4. PMUpdate Processing Rules .................................. 23
3.10. Operation: PMGetStatus .......................................... 24
  3.10.1. ws:Action values for PMGetStatus Messages ............... 24
  3.10.2. PMGetStatus Message ....................................... 24
  3.10.3. PMGetStatusResponse Message ............................... 25
  3.10.4. PMGetStatus Processing Rules ............................... 26
3.11. Operation: PMSetStatus .......................................... 26
  3.11.1. ws:Action values for PMSetStatus Messages ............... 26
  3.11.2. PMSetStatus Message ....................................... 27
  3.11.3. PMSetStatusResponse Message ............................... 27
  3.11.4. PMSetStatus Processing Rules ............................... 28
4. Provisioned Module Manager Service Schema .......................... 29
5. Provisioned Module Manager Service WSDL ................................ 34
References ............................................................................. 37
1. Introduction

The Provisioned Module Manager (PMM) is a component used to instantiate and manage Provisioned Modules. This specification documents the interfaces exposed by the PMM. The Provisioning Service Specification (see [LibertyPROV]) provides an overview of the provisioning process and describes how the PMM fits into this process.

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 pmm: represents the PMM namespace. This namespace is the default for instance fragments, type names, and element names in this document. In schema listings, and in examples of Discovery Service messages and fragments thereof, this is the default namespace when no prefix is shown:

  urn:liberty:pmm:2007-09

• The prefix prov: represents the Liberty ID-WSF Provisioning Service namespace [LibertyPROV]:

  urn:liberty:prov: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 Provisioned Module Manager (PMM) is a component in a system designed to enable run-time provisioning of Provisioned Modules (PM) – functionality modules (executables in some operating environments) to advanced client systems. The Liberty Advanced Client Technologies Overview (see [LibertyACT]) contains a complete description of how the entire system works together. The reader is strongly encouraged to review that document prior to digging into this specification.

The provisioning workflow provides for the instantiation of new PM(s) in a run-time environment over the wire. The PMM takes a critical role in this process as a beach head on the target platform which is able to instantiate the PM.

The PMM is frequently instantiated within some trusted environment which opens the potential for distributing PMs that contain functionality requiring a high level of trust and tamper resistance. We refer to such a PM as a Trusted Module (TM). For example, the TM may provide some of the IdP extension functionality described within the Liberty Advanced Client Technologies Overview document (see [LibertyACT]).

2.1. Provisioning Components

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

![Diagram of Provisioning Components]

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 beachhead 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. Provisioned Module Manager Service (PMM)

The Provisioned Module Manager Service (PMM) provides the interfaces used by Advanced Clients and/or ProvSs to initialize and/or maintain PMs.

An abstract WSDL definition for the PMM is included in this document, see Section 5: Provisioned Module Manager Service WSDL. This WSDL document defines all of the "WSDL operations" for the IdP Service.

The complete schema for the PMM is included in this document, see Section 4: Provisioned Module Manager Service Schema.

3.1. Service URIs

<table>
<thead>
<tr>
<th>Use</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Type</td>
<td>urn:liberty:pmm:2007-09</td>
</tr>
<tr>
<td>PMDeactivate wsa:Action</td>
<td>urn:liberty:pmm:2007-09:PMDeactivate</td>
</tr>
</tbody>
</table>

3.2. Status Codes

The following status code strings are defined:

- **OK**: message processing succeeded
- **Failed**: general failure code
- **Forbidden**: action by invoker is forbidden.
- **NotFound**: the requested object is not present.
- **Duplicate**: the object already exists or the task has already been accomplished.
3.3. Data Definitions

The PMM makes extensive use of the data elements defined in the Liberty ID-WSF Provisioning Service Specification [LibertyPROV] and does not define any of its own data element structures.

3.4. Request and Response Abstract Types

3.4.1. 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 PMM 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:

3.4.2. 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 definition of status codes are described in Section 3.2.

- anyAttribute [Optional] An attribute from a namespace other than that of this specification.
The following schema fragment defines the XML `ResponseAbstractType` complex type:

```xml
<!-- 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>
```

### 3.5. Operation: Provision

The provision operation is used by a local application on the Advanced client or a ProvS to instigate the provisioning of a new PM through the PMM.

When invoked by a local application on the Advanced Client, a `<prov:ProvisioningHandle>` is provided which provides the necessary information for the PMM to communicate with the ProvS (essentially an indirect provisioning process). When invoked by a ProvS, a `<PMDescriptor>` is typically provided enabling a direct provisioning process (though it is possible for the ProvS to use a `<prov:ProvisioningHandle>` when provisioning).

#### 3.5.1. wsa:Action values for Provision Messages

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

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

#### 3.5.2. Provision Message

The `<Provision>` request is called to provide the PMM with a `<prov:PMDescriptor>` or a `<prov:ProvisioningHandle>` (which can be resolved into a `<prov:PMDescriptor>`) instigating the provisioning of a new PM at the PMM.

The `<pmm:Provision>` request contains the following attributes and/or elements:

- `<prov:ProvisioningHandle>` [Optional] - the provisioning handle for the PM that is to be provisioned. This is the option that is typically specified by an untrusted application running on the same platform.
- `<prov:PMDescriptor>` [Optional] - the provisioning descriptor for the PM that is to be provisioned. This option is typically used by the ProvS when it is able to directly provision a PM to the PMM.
- `<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 for a delayed or proxied operation that completes at some point in the future.
- `wait` [Optional] - a boolean attribute indicating whether or not the caller wants to wait until the provisioning process for the PM is complete (the PM is initialized).

If not specified, or if the attribute is set to `true`, the PMM SHOULD wait for the provisioning to be complete.

If the attribute is set to `false`, the PMM SHOULD return an OK indicating an acknowledgement of the provisioning request as soon as the PMM is satisfied as to the validity of the request and the intent to proceed with the provisioning process.
• 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 invoker MUST specify the <prov:ProvisioningHandle> or the <prov:PMDescriptor> element. Both elements MUST NOT be specified.

The schema for the <pmm:Provision> is shown below.

An example message body containing a <pmm:Provision> message follows. This request presents a provisioning handle to the PMM to initiate the Provisioning process.

Example 1. Example <pmm:Provision> Message

3.5.3. ProvisionResponse Message

This response to the <pmm:Provision> request contains the following elements:
325  • <lu:Status>: [Required] - the status of the response. See the processing rules below for more information.
326  • anyAttribute [Optional] - zero or more attributes from a namespace other than that of this specification. One
327  such possibility is an xs:ID attribute such as xml:id or wsu:Id.
328
329  <!-- ProvisionResponse - response to the Provision request -->
330  <xs:element name="ProvisionResponse" type="ProvisionResponseType"/>
331  <xs:complexType name="ProvisionResponseType">
332     <xs:complexContent>
333         <xs:extension base="ResponseAbstractType" />
334     </xs:complexContent>
335  </xs:complexType>

338  Figure 3.  <pmm:ProvisionResponse> — Schema Fragment

339  An example message body containing a <pmm:ProvisionResponse> message follows. This is a successful
340  response.

341  <pmm:ProvisionResponse>
342      <lu:Status code="OK" />
343  </pmm:ProvisionResponse>

345  Example 2.  Example <pmm:ProvisionResponse> Message

346  3.5.4. Provision Processing Rules

347  • This operation adopts the Delayed Notification design pattern (see [LibertyDP]) and incorporates all of the
348  associated processing rules. Delayed notifications are needed for delayed operations (where the at attribute
349  is used with a time in the future).
350  Delayed Notifications SHOULD NOT to be used for immediate operations (no future at attribute). In such cases,
351  the PMM SHOULD wait for the completion of the request to return the appropriate status in its response to the
352  invoker.
353  If any portion of the request IS NOT an immediate operation and the <dp:NotifyTo> element IS NOT present
354  on the request, the PMM should validate the request to the extent possible and respond with the results of that
355  validation. If that validation succeeded, the PMM MUST continue with the execution of the operation. In such
356  cases the actual completion status of the operation may be different from what was reported (e.g. the request may
357  fail at the time it is attempted for some reason), but that will not be known to the invoker given that they have
358  chosen to not provide a delayed notification endpoint.
359  If any portion of the request IS NOT an immediate operation and the <dp:NotifyTo> element IS present on the
360  request, the PMM MUST report the final completion status (e.g. wait for the completion status of any delayed or
361  indirect operation at the PMM) of the request. If multiple operations are included in the request, the ProvS MAY
362  group the results of some or all of the operations in a single delayed notification message.
363  In the case of a mixed request, the immediate portion of the request SHOULD be handled as an immediate request
364  and the actual results returned with the initial response.
365  • If the wait attribute is set to a boolean true value, the PMM MUST wait until the provisioning process is complete
366  prior to sending the response to this request. This, potentially, includes the steps of obtaining the PMDescriptor
367  and the PMEngine from the ProvS.

Liberty Alliance Project

11
• If data in the input parameters is invalid (such as a missing PMArtifact in the PH) the PMM MUST treat the request as a failure. The PMM SHOULD return an unsuccessful status in the response. If detailed status codes are being included, the detailed status code for this error MUST be "Invalid."

If the PMM is unable to return an unsuccessful status in the response, it MUST, instead, generate a SOAP fault message. This allows for tooling situations where the request format error is processed by the SOAP/XML tooling layer prior to getting to the PMM instance.

• If the input parameters require a capability that is unsupported within the PMM (such as an unsupported Security Mechanism or framework version) such that it is impossible for the PMM to complete the task, the PMM MUST treat the request as a failure. If detailed status codes are being included, the detailed status code for this error MUST be "Unsupported."

• If request processing succeeded, the top-level status code MUST be "OK." If the request has been accepted and is subject to delayed notification, the top-level status code MUST be "WillNotify." Otherwise, the top-level status code MUST be "Failed."

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

3.6. Operation: PMActivate

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

3.6.1. wsa:Action values for PMActivate Messages

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

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

3.6.2. PMActivate Message

The <PMActivate> request is called by a provisioning entity (such as the Liberty ID-WSF Provisioning Service - see [LibertyPROV]) to activate an inactive PM at the PMM.

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

• <pmm:PMActivateItem> [Required] - one or more activate request items which contain the following elements/attributes:
  • <PMID> [Required] - the identifier of the PM being activated.
  • itemID [Required] - a unique (in this request) identifier for this <pmm:PMActivateItem> element. This value is used to correlate the response data to this request data.
  • 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 PMM would activate the PM.
    If this attribute is not specified, or it is specified with a time in the past, the PM MUST be activated now.
• <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 for a delayed or proxied operation that completes at some point in the future.

• 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 <pmm:PMActivate> is shown below.

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

Example 3. Example <pmm:PMActivate> Message
3.6.3. PMActivateResponse Message

This response to the <pmm: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.

```xml
<!-- PMActivateResponse - the response to the PMActivate request -->
<pmm:PMActivateResponse type="PMActivateResponseType"/>
<x:complexType name="PMActivateResponseType">
    <x:complexContent>
        <x:extension base="ResponseAbstractType" />
    </x:complexContent>
</x:complexType>
```

Figure 5. <pmm:PMActivateResponse> — Schema Fragment

An example message body containing a <pmm:PMActivateResponse> message follows. This is a partial response. The 2nd activation, which is an immediate operation, is successful. The 1st activation is queued for processing and the PMM will notify the invoker when it completes.

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

Example 4. Example <pmm:PMActivateResponse> Message

3.6.4. 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 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). In such cases, the PMM SHOULD wait for the completion of the request to 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 PMM should validate the request to the extent possible and respond with the results of that validation. If that validation succeeded, the PMM 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 time it is attempted 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 PMM 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 PMM MUST result in a failure. If detailed status codes are being included in the response, the detailed status code for this error MUST be "NotFound."

• Requests to activate a PM that is already activated MUST result in a failure unless the request is for a future time when the PMM can determine that the PM will be inactive at that point in time. If detailed status codes are being included in the response, the detailed status code for this error MUST be "AlreadySo."

• If the timestamp specified in the at attribute is in the future and exactly matches the future timestamp for a pending deactivation, this request MUST have the effect of canceling the pending request (such that neither of the two requests remain pending). This is the only means to cancel a pending deactivation request.

In such cases, if the cancellation is successful, this request should return a successful status.

• If a pending activation is canceled (via a <pmm:PMDeactivate> with the exact same at attribute value) and delayed notification is in effect for the activation, the completion status included in the notification message MUST be a failure. If detailed status codes are included, the detailed status code for this case MUST be "Canceled."

• The behavior of the PMM when multiple pending occurrences of the same operation are requested (such as multiple future activations) is NOT defined by this specification. It is recommended that callers NOT attempt such operations. If an implementation of a PMM chooses to refuse such duplicated operations and treat them as a failure, the detailed status code, if present, MUST be "Duplicate."

• If the activation is to take place immediately (e.g., there was no at attribute specified), the PMM SHOULD wait until the activation process is complete prior to sending the response to this request.

• If this operation is to take place at some point in the future, the PMM SHOULD, to the extent possible, validate the parameters of the request to detect any errors. Any such errors detected MUST be reported on the response to this request.

• 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 PMM wants to share that information with the invoking party.

3.7. Operation: PMDeactivate

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

3.7.1. wsa:Action values for PMDeactivate Messages

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

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

3.7.2. PMDeactivate Message

The <PMDeactivate> request is called by a provisioning entity (such as the Liberty ID-WSF Provisioning Service - see [LibertyPROV]) to deactivate an active PM at the PMM.

The <pmm:PMDeactivate> request contains the following attributes and/or elements:
<pmm:PMDeactivateItem> [Required] - one or more deactivate request items which contain the following elements/attributes:

  • <PMID> [Required] - the identifier of the PM being deactivated.

  • itemID [Required] - a unique (in this request) identifier for this <pmm:PMDeactivateItem> element. This value is used to correlate the response data to this request data.

  • 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 PMM would deactivate the PM.

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

  • <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 for a delayed or proxied operation that completes at some point in the future.

  • 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 <pmm:PMDeactivate> is shown below.

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

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

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

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

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

Figure 6. <pmm:PMDeactivate> — Schema Fragment

An example message body containing a <pmm:PMDeactivate> message follows. This request deactivates two PMs, one now and one near midnight New Year’s eve.

```
<pmm:PMDeactivate itemID="1" at="2007-01-31T23:59:59Z">
</pmm:PMDeactivateItem>
<pmm:PMDeactivate itemID="2">
</pmm:PMDeactivateItem>
```
3.7.3. PMDeactivateResponse Message

This response to the <pmm: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`.

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

Example 6. Example `<pmm:PMDeactivateResponse>` Message

3.7.4. PMDeactivate Processing Rules

- `<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 for a delayed or proxied operation that completes at some point in the future.
- If a specified PMID is not known to the PMM, the PMM 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."
- If the deactivation is to take place immediately (e.g., there was no `at` attribute specified), the PMM SHOULD wait until the deactivation process is complete prior to sending the response to this request.
- Requests to deactivate a PM that is not active MUST result in a failure unless the request is for a future time when the PMM can determine that the PM will be active at that point in time. If detailed status codes are being included in the response, the detailed status code for this error MUST be "AlreadySo."
- If this operation is to take place at some point in the future, the PMM SHOULD, to the extent possible, validate the parameters of the request to detect any errors. Any such errors detected MUST be reported on the response to this request.
• If the timestamp specified in the at attribute is in the future and exactly matches the future timestamp for a pending activation, this request MUST have the effect of canceling the pending request (such that neither request remains pending). This is the only means to cancel a pending activation request.

In such cases, the deactivation request is considered successful (and returns a successful response) but has no effect other than to cancel the pending activation.

• If a pending deactivation is canceled (via a <pmm:PMActivate> with the exact same at attribute value) and delayed notification is in effect for the deactivation, the completion status in the notification message MUST indicate a failure. If detailed status codes are included, the detailed status code for this case MUST be "Canceled."

• The behavior of the PMM when multiple pending occurrences of the same operation are requested (such as multiple future deactivations) is NOT defined by this specification. It is recommended that callers NOT attempt such operations. If an implementation of a PMM chooses to refuse such duplicated operations and treat them as a failure, the detailed status code, if present, MUST be "Duplicate."

• 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 PMM wants to share that information with the invoking party.

3.8. Operation: PMDelete

The PMDelete operation is used by a provisioning entity to delete a currently installed PM at the PMM.

3.8.1. wsa:Action values for PMDelete Messages

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

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

3.8.2. PMDelete Message

The <PMDelete> request is called by a provisioning entity (such as the Liberty ID-WSF Provisioning Service - see [LibertyPROV]) to delete a PM at the PMM.

If the PM is active at the time of deletion, the PM is deactivated and deleted in a single operation (in other words, there is no need to call <pmm:PMDeactivate> prior to calling <pmm:PMDelete>).

The <pmm:PMDelete> the following attributes and/or elements:

• <pmm:PMDeleteItem> [Required] - one or more containing the following elements and/or attributes:

  • <PMID> [Required] - the identifier of the PM being deleted.

  • itemID [Required] - a unique (in this request) identifier for this <pmm:PMDeleteItem> element. This value is used to correlate the response data to this request data.

  • 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 `<pmm:PMDelete>` is shown below.

```xml
<!-- PMDelete - to delete a PM at the PMM -->
<pmm:PMDelete type="PMDeleteType"/>

<xsd:complexType name="PMDeleteType">
  <xsd:complexContent>
    <xsd:extension base="RequestAbstractType">
      <xsd:sequence>
        <xsd:element ref="PMDeleteItem" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="PMDeleteItem" type="PMDeleteItemType"/>

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

Figure 8. `<pmm:PMDelete>` — Schema Fragment

An example message body containing a `<pmm:PMDelete>` message follows. This request deletes one PM.

```xml
<pmm:PMDelete itemID="1">
</pmm:PMDeleteItem>
</pmm:PMDelete>
```

Example 7. Example `<pmm:PMDelete>` Message

### 3.8.3. PMDeleteResponse Message

This response to the `<pmm:PMDelete>` 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`.
<xs:element name="PMDeleteResponse" type="PMDeleteResponseType"/>
<xs:complexType name="PMDeleteResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>

Figure 9. `<pmm:PMDeleteResponse>` — Schema Fragment

An example message body containing a `<pmm:PMDeleteResponse>` message follows. This is a successful response.

Example 8. Example `<pmm:PMDeleteResponse>` Message

3.8.4. PMDelete Processing Rules

- If a specified PMID is not known to the PMM, or has already been deleted, the PMM 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."

- All pending operations (such as a pending future deactivation) related to this PM MUST be canceled. If delayed notification is in effect for any of the pending operations, the completion status included in the notification message MUST indicate failure. If detailed status codes are included in the notification, the detailed status code for this case MUST be "Canceled."

- If request processing succeeded for all PMs, the top-level status code MUST be `OK`. If the request processing failed for all PMs, the top-level status code MUST be `Failed`. Otherwise, if the results were mixed, the top-level status MUST be `Partial`, and a second-level status MUST be included for the items for which the processing was not successful. The second level status for such items MUST indicate that the processing failed and MUST include 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 is not `OK` and second level status codes are present, they MAY contain detailed error information if the PMM wants to share that information with the invoking party.
3.9. Operation: **PMUpdate**

The **PMUpdate** operation is used by a provisioning entity to update a currently installed PM at the PMM.

3.9.1. **wsa:**Action values for **PMUpdate** Messages

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

3.9.2. **PMUpdate** Message

The `<PMUpdate>` request is called by a provisioning entity (such as the Liberty ID-WSF Provisioning Service - see [LibertyPROV]) to update one or more PM(s) at the PMM.

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

- `<pmm:PMUpdateItem>` **[Required]** - one or more update request items which contain the following elements/attributes:
  - `<prov:PMDescriptor>` **[Required]** - the replacement/updated information for the PM that is to be updated. The contents of this element will vary depending upon the setting of the `type` attribute.
  - The interpretation of the contents within the `<prov:PMDescriptor>` is similar to the interpretation of those same elements when processing an initial provisioning operation including:
    - The `<PMID>` element identifies the existing PM that is to be updated.
    - 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.
    - The `<prov:PMEngineRef>`, `<prov:PMInitData>`, and `<prov:PMRTData>` elements all contain the updated information, if any, for this update operation.
    - `type` **[Required]** - the type of update being requested. This attribute has the same values and interpretations as the `type` attribute used on the `<prov:PMDUpdateItem>` in the `<prov:PMDUpdate>` interface in the Liberty Provisioning Service Specification ([LibertyPROV]).
    - `itemID` **[Required]** - a unique (in this request) identifier for this `<pmm:PMUpdateItem>` element. This value is used to correlate the response data to this request data.
    - `at` **[Optional]** - an optional time at which the update should take place. If specified, this SHOULD be some time in the future. If this attribute is not specified, or it is specified with a time in the past, the PM MUST be updated now.
    - `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`.
  - `<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 for a delayed or proxied operation that completes at some point in the future.
  - `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 `<pmm:PMUpdate>` is shown below.

```
<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="prov: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 10. `<pmm:PMUpdate>` — Schema Fragment

An example message body containing a `<pmm:PMUpdate>` message follows. This request updates the engine only of a single PM.

```
<pmm:PMUpdate>
  <pmm:PMUpdateItem itemID="1" type="urn:liberty:prov:2007-09:ut:engine">
    <prov:PMDescriptor xs:id="2323923900239">
      <prov:PMEngineRef>https://pmsRus.org/VeryTrustedModule/4.0</prov:PMEngineRef>
      <ds:Signature>
        ... signature data goes here ...
      </ds:Signature>
    </prov:PMDescriptor>
  </pmm:PMUpdateItem>
</pmm:PMUpdate>
```

Example 9. Example `<pmm:PMUpdate>` Message

### 3.9.3. PMUpdateResponse Message

This response to the `<pmm:PMUpdate>` 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`. 
<xs:element name="PMUpdateResponse" type="PMUpdateResponseType"/>

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

Figure 11. <pmm:PMUpdateResponse> — Schema Fragment

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

Example 10. Example <pmm:PMUpdateResponse> Message

If delayed notifications are in effect for the update, the PMM would need to subsequently send a notification message with the completion status for the update. The example below shows such a notification.

Example 11. Example <pmm:PMUpdateResponse> Message in a notification

3.9.4. 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 delayed operations (where the at attribute is used with a time in the future).

Delayed Notifications SHOULD NOT be used for immediate operations (no future at attribute). In such cases, the PMM SHOULD wait for the completion of the request to 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 PMM should validate the request to the extent possible and respond with the results of that validation. If that validation succeeded, the PMM 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 time it is attempted 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 PMM 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.
• If a specified PMID is not known to the PMM, the PMM 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."

• Requests that include invalid parameters, such as an invalid or missing type attribute, MUST result in a failure. If detailed status codes are being included in the response, the detailed status code for this error MUST be "Invalid."

• If the update is to take place immediately (e.g., there was no at attribute specified), the PMM SHOULD wait until the update process is complete prior to sending the response to this request.

• A cancellation request for a PM which has NO pending update scheduled to take place at the time specified in the at attribute MUST result in a failure. If detailed status codes are included in the response, the detailed status code for this case MUST be "NotFound."

• If a pending update is canceled (via a <pmm:PMUpdate> with the exact same at attribute value and an update type of "urn:liberty:prov:2007-09:ut:cancel") and delayed notification is in effect for the update, the completion status included in the notification MUST indicate a failure. If detailed status codes are included in the notification, the detailed status code for this case MUST be "Canceled."

• If this operation is to take place at some point in the future, the PMM SHOULD, to the extent possible, validate the parameters of the request to detect any errors. Any such errors detected MUST be reported on the response to this request.

• 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.

• The behavior of the PMM when multiple pending occurrences of the same operation are requested (such as multiple future updates) is NOT defined by this specification. It is recommended that callers NOT attempt such operations. If an implementation of a PMM chooses to refuse such duplicated operations and treat them as a failure, the detailed status code, if present, MUST be "Duplicate."

• 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 PMM wants to share that information with the invoking party.

3.10. Operation: PMGetStatus

The PMGetStatus operation is used by a provisioning entity to get the current status of a PM at the PMM.

3.10.1. wsa:Action values for PMGetStatus Messages

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

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

3.10.2. PMGetStatus Message

The <PMGetStatus> request is called by a provisioning entity (such as the Liberty ID-WSF Provisioning Service - see [LibertyPROV]) to obtain the current status of one or more PM(s) at the PMM.
The `<pmm:PMGetStatus>` request contains the following attributes and/or elements:

- `<prov:PMID>` [Optional] - zero or more PM identifiers for the PMs whose status is being requested. If no PMs are identified, the request is for the status of all PMs managed by the invoker (typically the ProvS from which the PM was provisioned).
- `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 `<pmm:PMGetStatus>` is shown below.

An example message body containing a `<pmm:PMGetStatus>` message follows. This requests the status of a single PM.

```
<pmm:PMGetStatus>
</pmm:PMGetStatus>
```

Example 12. Example `<pmm:PMGetStatus>` Message

### 3.10.3. PMGetStatusResponse Message

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

- `<lu:Status>` [Required] - the status of the response. See the processing rules below for more information.
- `<prov:PMStatus>` [Optional] - zero or more PM status elements describing the current status of the PM. This element MUST be included for any PM that matches the PMID(s) specified in the request.
- `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="prov:PMStatus" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

Figure 13. `<pmm:PMGetStatusResponse>` — Schema Fragment

An example message body containing a `<pmm:PMGetStatusResponse>` message follows. This is a successful response.

```xml
<pmm:PMGetStatusResponse>
  <lu:Status code="OK" />
  <prov:PMStatus>
  </prov:PMStatus>
</pmm:PMGetStatusResponse>
```

Example 13. Example `<pmm:PMGetStatusResponse>` Message

### 3.10.4. PMGetStatus Processing Rules

- The invoking party SHOULD specify a value for the `<PMID>` elements that match the value of existing, provisioned PMs. If this is not the case, the PMM MUST ignore such values and treat the request as if they were not present on the request.

- When this interface is used by a party other than the PMD issuing authority or the ProvS which provisioned the PM, the PMM MUST treat the request as if the PM does not exist.

- 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 PMs, the top-level status code MUST be `OK`. If the request processing failed for all PMs, 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 `<pmm:PMStatus>` element(s) for the PMs 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 PMM wants to share that information with the invoking party.
3.11. Operation: PMSetStatus

The PMSetStatus operation is used by the PM to set the current status of the PM at the PMM.

3.11.1. wsa:Action values for PMSetStatus Messages

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

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

3.11.2. PMSetStatus Message

The PMSetStatus request is called by a PM to set the current status of the PM at the PMM.

The <pmm:PMSetStatus> request contains the following attributes and/or elements:

- <prov:PMStatus> [Required] - the updated status information for the PM.
- 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 <pmm:PMSetStatus> is shown below.

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

An example message body containing a <pmm:PMSetStatus> message follows.

```
<pmm:PMSetStatus>
    <prov:PMStatus>
    </prov:PMStatus>
</pmm:PMSetStatus>
```

3.11.3. PMSetStatusResponse Message

This response to the <pmm: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.

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

Figure 15. <pmm:PMSetStatusResponse> — Schema Fragment

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

3.11.4. PMSetStatus Processing Rules

• The PMM MUST ensure that only the provisioned PM is allowed to use this interface to directly change the status of the PM. Invocations of this interface by any other party MUST result in a failure.

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 PMM MUST ignore this value and, instead, use its understanding of the current time to record as the status time.

If the <State> element within the <PMStatus> contains a value that is not understood by the PMM, the PMM MAY treat the operation as a failure. PMMs MUST accept all of the Liberty defined status values specified in [LibertyPROV].

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. If the request processing failed, the top-level status code MUST be Failed.

If the top-level status is not OK, and second level status codes are present, they MAY contain detailed error information if the PMM wants to share that information with the invoking party.
4. Provisioned Module Manager Service Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:pmm: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:pmm="urn:liberty:pmm:2007-09"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <!-- 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>
  <!-- Provision - to instigate the provisioning of a PM at the PMM -->
  <xs:element name="Provision" type="ProvisionType"/>
  <xs:complexType name="ProvisionType">
    <xs:complexContent>
      <xs:extension base="RequestAbstractType">
        <xs:sequence>
          <xs:choice>
            <xs:element ref="prov:ProvisioningHandle" />
            <xs:element ref="prov:PMDescriptor" />
          </xs:choice>
          <xs:element ref="dp:NotifyTo" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="wait" type="xs:boolean" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- ProvisionResponse - response to the Provision request -->
  <xs:element name="ProvisionResponse" type="ProvisionResponseType"/>
  <xs:complexType name="ProvisionResponseType">
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="prov:ProvisioningHandle"/>
        <xs:element ref="prov:PMDescriptor"/>
      </xs:sequence>
      <xs:attribute name="wait" type="xs:boolean" use="optional"/>
    </xs:extension>
  </xs:complexType>
</xs:schema>
```
<xs:complexContent>
  <xs:extension base="ResponseAbstractType" />
</xs:complexContent>

<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>

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

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

<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>
<xs:element name="PMDeactivateResponse" type="PMDeactivateResponseType"/>
<xs:complexType name="PMDeactivateResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType"/>
  </xs:complexContent>
</xs:complexType>

<!-- PMDelete - to delete a PM at the PMM -->
<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: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>

<!-- PMUpdate - update the PM at the PMM -->
<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:sequence>
</xs:complexType>
<xs:element ref="prov: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>

<!-- PMGetStatus - to check the provisioning status of a PM at the PMM -->
<xs:element name="PMGetStatus" type="PMGetStatusType"/>
<xs:complexType name="PMGetStatusType">
<xs:complexContent>
<xs:extension base="RequestAbstractType">
<xs:sequence>
<xs:element ref="prov: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="prov:PMStatus" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- PMSetStatus - to update the status of a PM at the PMM -->
<xs:element name="PMSetStatus" type="PMSetStatusType"/>
<xs:complexType name="PMSetStatusType">
<xs:complexContent>
<xs:extension base="RequestAbstractType">
<xs:sequence>
<xs:element ref="prov:PMStatus"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- PMSetStatusResponse - response to the PMSetStatus request -->
<xs:element name="PMSetStatusResponse" type="PMSetStatusResponseType"/>
<xs:complexType name="PMSetStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType"/>
  </xs:complexContent>
</xs:complexType>
5. Provisioned Module Manager Service WSDL

```xml
<?xml version="1.0"?>
<definitions name="pmm-svc"

targetNamespace="urn:liberty:pmm:2007-09"
xmlns:tns="urn:liberty:pmm:2007-09"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/"
xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsd1"
xmlns:pmm="urn:liberty:pmm:2007-09"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

<message name="Provision">
<part name="body" element="pmm:Provision"/>
</message>

<message name="ProvisionResponse">
<part name="body" element="pmm:ProvisionResponse"/>
</message>

<message name="PMActivate">
<part name="body" element="pmm:PMActivate"/>
</message>

<message name="PMActivateResponse">
<part name="body" element="pmm:PMActivateResponse"/>
</message>

<message name="PMDeactivate">
<part name="body" element="pmm:PMDeactivate"/>
</message>

<message name="PMDeactivateResponse">
<part name="body" element="pmm:PMDeactivateResponse"/>
</message>

<message name="PMDelete">
<part name="body" element="pmm:PMDelete"/>
</message>

<message name="PMDeleteResponse">
<part name="body" element="pmm:PMDeleteResponse"/>
</message>

<message name="PMGetStatus">
<part name="body" element="pmm:PMGetStatus"/>
</message>

<message name="PMGetStatusResponse">
<part name="body" element="pmm:PMGetStatusResponse"/>
</message>

<message name="PMSetStatus">
<part name="body" element="pmm:PMSetStatus"/>
</message>

<message name="PMSetStatusResponse">
<part name="body" element="pmm:PMSetStatusResponse"/>
</message>

<message name="PMUpdate">
<part name="body" element="pmm:PMUpdate"/>
</message>
```

<part name="body" element="pmm:PMUpdate"/>
</message>
<message name="PMUpdateResponse">
  <part name="body" element="pmm:PMUpdateResponse"/>
</message>

<portType name="PMMPort">
  <operation name="Provision">
    <input message="tns:Provision"
    <output message="tns:ProvisionResponse"
  </operation>
  <operation name="PMActivate">
    <input message="tns:PMActivate"
    <output message="tns:PMActivateResponse"
  </operation>
  <operation name="PMDeactivate">
    <input message="tns:PMDeactivate"
    <output message="tns:PMDeactivateResponse"
  </operation>
  <operation name="PMDelete">
    <input message="tns:PMDelete"
    <output message="tns:PMDeleteResponse"
  </operation>
  <operation name="PMGetStatus">
    <input message="tns:PMGetStatus"
    <output message="tns:PMGetStatusResponse"
  </operation>
  <operation name="PMSetStatus">
    <input message="tns:PMSetStatus"
    <output message="tns:PMSetStatusResponse"
  </operation>
  <operation name="PMUpdate">
    <input message="tns:PMUpdate"
    <output message="tns:PMUpdateResponse"
  </operation>
</portType>

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

<message name="PMMBinding" type="tns:PMMPort">
  <soap:binding style="document" />
<operation name="Provision">
    <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="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="PMUpdate">
    <input> <soap:body use="literal"/> </input>
    <output> <soap:body use="literal"/> </output>
</operation>
References

Normative


Liberty Alliance Project: Liberty ID-WSF Provisioned Module Manager Service Specification


Informative


