**The Policy Machine**

**Installation Guide**

**Version 2**

##### By: Serban I. Gavrila

**Updated by: Nicole Keller**

Created December 5, 2006

Updated: 2013

[1 System Requirements 3](#_Toc153341481)

[1.1 Hardware Requirements 3](#_Toc153341482)

[1.1.1 Hardware Requirements for the Policy Machine Server 3](#_Toc153341483)

[1.1.2 Hardware Requirements for the Target Systems 3](#_Toc153341484)

[1.2 Software Requirements 3](#_Toc153341485)

[1.2.1 Software Requirements for the Policy Machine Server: 3](#_Toc153341486)

[1.2.2 Software Requirements for the Target Systems 3](#_Toc153341487)

[2 Installing the Policy Machine Engine 3](#_Toc153341488)

[2.1 Overview 3](#_Toc153341489)

[2.2 Extending the Active Directory Schema 5](#_Toc153341490)

[2.3 Creating Policy Machine Data Containers 5](#_Toc153341491)

[2.4 Obtaining the Policy Machine Server Certificate 7](#_Toc153341492)

[2.5 Running the Policy Machine Server 9](#_Toc153341493)

[3 Installing the Policy Machine Administrative Tool 9](#_Toc153341494)

[3.1 Overview 9](#_Toc153341495)

[3.2 Obtaining User Certificates 10](#_Toc153341496)

[3.3 Running the Policy Machine Administrative Tool 11](#_Toc153341497)

[4 Installing the User Environment Simulator 12](#_Toc153341498)

[4.1 Overview 12](#_Toc153341499)

[4.2 Running the User Environment Simulator 12](#_Toc153341500)

[5 Installing the Applications 13](#_Toc153341501)

# 1 System Requirements

## Hardware Requirements

### 1.1.1 Hardware Requirements for the Policy Machine Server

* At least 1 GHz Pentium CPU (at least 2 GHz recommended).
* Minimum 256 MB of RAM (512 MB recommended).
* A hard disk partition with enough space to accommodate the Policy Machine Server code – approx. 150KB.
* Adequate space in the Active Directory for the Policy Machine data.
* VGA or higher-resolution monitor.
* Keyboard.
* Mouse.
* Network adapter.

### 1.1.2 Hardware Requirements for the Target Systems

* At least 400 MHz CPU.
* Minimum 128 MB of RAM.
* A hard disk partition with enough space to accommodate the User Environment Simulator code – approx. 100KB.
* VGA or higher-resolution monitor.
* Keyboard.
* Mouse.
* Network adapter.

## Software Requirements

### 1.2.1 Software Requirements for the Policy Machine Server:

* Windows 2000 Server or Windows 2003 Server with DNS and Active Directory installed.
* MS Certificate Services installed.
* Java Standard Edition Runtime Environment (JRE) 1.4.1 or newer.

### 1.2.2 Software Requirements for the Target Systems

* Windows 2000 or XP
* Java Standard Edition Runtime Environment (JRE) 1.4.1 or newer.

**1.2.3 Software requirements for PM Applications**

- JavaMail

- OpenOffice both the SDK and the EXE

- Apache Ant 1.9.0

## 2 Installing the Policy Machine Engine

## 2.1 Overview

The Policy Machine is distributed as a ZIP file with the following contents:

* server.jar: the code of the Policy Machine (PM) server.
* admin.jar: the code for the PM Administrative Tool.
* simulator.jar: the code of the User Environment Simulator.
* editor.jar: the code of a PM application (a Rich Text Format editor).
* thEditor.jar: the code of a PM application (a Rich Text Format editor) that contains a Trojan horse.
* wkf.jar: the code of a PM application (a workflow application).
* Dot.exe: the AT&T’s tool for generating graph images from textual descriptions.
* PMAttributes.ldf: a text file containing the definitions of PM-specific Active Directory attributes.
* PMClasses.ldf: a text file containing the definitions of PM-specific Active Directory classes.
* PMInstallGuide.doc: this guide as a MS Word document.
* PMUserGuide.doc: the PM User Guide as a MS Word document.

To install the PM engine (server), follow these steps:

Step 1. Log on as Administrator on the computer that will host the PM server. Please see the hardware and software requirements in the previous sections.

Step 2. Unpack the distribution file to a folder of your choice on the computer that will host the PM server.

Step 3. Use the files PMAttributes.ldf and PMClasses.ldf to extend the Active Directory Schema according to the procedure outlined in section 2.2.

Step 4. Create the Active Directory containers for the PM data, as detailed in Section 2.3.

Step 5. Obtain a X.509 certificate for the PM Server and store its private key and the certificate into a key store. Store the root certificate(s) in a trust store. Section 2.4 describes the procedure that uses a MS Stand-Alone Certification Authority (CA), which can be found on the Windows 2000 or 2003 Servers if you installed MS Certification Services. Of course, the certificate can be obtained from other CAs as well.

To run the PM engine (server), use the MS-DOS command (everything on one line):

java -cp E:\PolicyMachine\PMServer\server.jar

-Djavax.net.ssl.keyStore=E:\PolicyMachine\Keystores\serverKeystore

-Djavax.net.ssl.keyStorePassword=aaaaaa

-Djavax.net.ssl.trustStore=E:\PolicyMachine\Keystores\serverTruststore

pmserver.PmEngine

We assumed that the server.jar file is stored in E:\PolicyMachine\PMServer\ directory, and that the server’s key store is E:\PolicyMachine\Keystores\serverKeystore, while the server’s trust store is E:\PolicyMachine\Keystores\serverTruststore. You need to update the command line according to your particular setup. Also, instead of the key store password “aaaaaa” use the password you set up when you created the key store.

## 2.2 Setting Up Active Directory and DNS Server

Step 1: Open the Server Manager

Step 2: Select action in the tool bar and then click add roles

Step 3: Add Active Directory Domain Services and DNS server there will be instructions on how to start the active directory wizard

At the end of the wizard you should have set up the users domain name to look like this : firstInitialLastNameLab.company.com, org, gov, etc.The forest functional level should be Server 2008 R2. The NETBIOS Name is usually the first part of your user domain name.

## 2.2 Extending the Active Directory Schema

The Active Directory Schema must be extended with a set of PM-specific AD attributes and classes. Appendix A contains a description of them.

You may extend the AD schema by following the steps:

Step 1. Log on as administrator on the Windows 2000 or 2003 server that will host the PM server.

Step 2. Add the following key to the registry:

HKEY\_LOCAL\_MACHINE/System/CurrentControlSet/Services/NTDS/Parameters/Schema Update Allowed

with the value (REG\_DWORD)1 (i.e., true).

Step 3. Update the two files PMAttributes.ldf and PMClasses.ldf, which were contained in the distribution file, to reflect your domain name. Specifically, replace all occurrences of the string “DC=pm1, DC=local” by your domain name, for example “DC=fabrikam, DC=com” in both files.

Step 4. Open a MS-DOS command window, cd to the directory that contains the two edited files PMAttributes.ldf, PMClasses.ldf, and PMContainers.ldf and run the commands:

ldifde -i -f PMAttributes.ldf

ldifde -i -f PMClasses.ldf

ldifde –i –f PMContainers.ldf

Afterwards, you may use the ADSI editor to examine your changes. Look for attributes and classes with the prefix “pm”.

Note that once a class is created, it cannot be modified or deleted (the only exception is adding or deleting *optional* attributes to/from the class).

## 2.3 Creating Policy Machine Data Containers

**NOTE: This isn’t needed if you already have loaded the PMContainers.ldf file. But I would advise you to look through the data below and make sure that all the containers, attributes and classes that are listed are in the ADSI editor.**

One must create containers for the PM data by following the steps:

Step 1. Log on as Administrator on the Windows 2000 or 2003 server that will host the PM server.

Step 2. Use the ADSI editor to create the following containers under the root domain level (i.e., DC=fabrikam, DC=com):

PmAdminVosNodeContainer

PmAttributeSetContainer

PmConnectorContainer

PmDenyContainer

PmEventContainer

PmHostContainer

PmNameContainer

PmObjectAttributeContainer

PmObjectClassContainer

PmOperationSetContainer

PmOsConfigContainer

PmPolicyContainer

PmRuleContainer

PmSacContainer

PmSconaContainer

PmSconContainer

PmSessionContainer

PmTaskContainer

PmUserAttributeContainer

PmUserConfigContainer

PmUserContainer

PmVirtualObjContainer

PmVosNodeContainer

Step 3. Using the ADSI editor, create the following objects of class pmClassNameMap in the container PmNameContainer:

###### Object name Value of the pmName attribute

ActionClassName pmClassAction

AdminVosNodeClassName pmClassAdminVosNode

AttributeSetClassName pmClassAttributeSet

ConditionClassName pmClassCondition

ConnectorClassName pmClassConnector

DenyClassName pmClassDeny

EventClassName pmClassEvent

EventPatternClassName pmClassEventPattern

HostClassName pmClassHost

ObjectAttributeClassName pmClassObjectAttribute

ObjectClassClassName pmClassObjectClass

OperandClassName pmClassOperand

OperationSetClassName pmClassOperationSet

OsConfigClassName pmClassOsConfig

PolicyClassName pmClassPolicy

RuleClassName pmClassRule

SacClassName pmClassSac

SacClassName pmClassSac

SconaClassName pmClassScona

ScriptClassName pmClassScript

ScriptSourceClassName pmClassScriptSource

SessionClassName pmClassSession

SourceLineClassName pmClassSourceLine

StaticConstraintClassName pmClassStaticConstraint

TaskClassName pmClassTask

UserAttributeClassName pmClassUserAttribute

UserClassName pmClassUser

UserConfigClassName pmClassUserConfig

VirtualObjectClassName pmClassVirtualObject

VosNodeClassName pmClassVosNode

Step 4. Using the ADSI editor’s Properties/Security tab pane, delete “Authenticated Users” from the list of users authorized to access the above containers.

## 2.4 Obtaining the Policy Machine Server Certificate

Clients like PM Administrative Tool communicate with the PM Engine (server) using SSL, hence the need for the server (and the client) to identify itself to the other party through public key certificates. This section explains how to obtain a server certificate from a MS Stand-alone Certification Authority (CA).

Step 1. Log as Administrator on the computer that will host the PM engine (server). Make sure that a stand-alone certification authority is installed, and that it automatically issues the requested certificates (in order to simplify the procedure). Install the CA root certificate in the Trusted Root Certification Authorities.

Step 2. Create a folder named Keystores. This folder will contain the PM server key store and trust store. The key store is a repository for the PM server’s private key and certificate. The trust store is a repository for public key certificates that the PM server trusts. For simplicity, we assume that the same CA is used to issue all certificates. Then the trust store needs only contain the CA’s root certificate.

Step 3. Open a MS-DOS command window, cd to the Keystores folder, and run the following commands:

>keytool -genkey -keyalg RSA -alias pmserver -keystore serverKeystore

Enter keystore password: aaaaaa

What is your first and last name?

[Unknown]: pmserver

What is the name of your organizational unit?

[Unknown]: CSD

What is the name of your organization?

[Unknown]: NIST

What is the name of your City or Locality?

[Unknown]: Gaithersburg

What is the name of your State or Province?

[Unknown]: Maryland

What is the two-letter country code for this unit?

[Unknown]: US

Is CN=pmserver, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US correct?

[no]: yes

Enter key password for <pmserver>

(RETURN if same as keystore password):

>keytool -certreq -alias pmserver -file pmserver.csr -keystore serverKeystore

Enter keystore password: aaaaaa

The file pmserver.csr contains a standard certificate request.

Step 4. Use the MS Internet Explorer to submit the certificate request to the CA installed on the MS Windows Server 2000 or 2003. Namely, open the URL

<http://mymachine/certsrv>

where “mymachine” must be replaced by the server’s host name. From the web page displayed, select the radio button “Request a certificate” and click the “Next” button. From the next page, select the radio button “Advanced request” and click the “Next” button. From the next page, select the radio button “Submit a certificate request using a base64 encoded PKCS #10 file or a renewal request using a base64 encoded PKCS #7 file” and click the “Next” button. Copy the contents of the pmserver.csr file and paste it in the text area titled “Saved Request”. Click the “Submit” button.

In the next page, select the radio button “Base 64 encoded” and select the link “Download CA certificate”. When asked, save the downloaded certificate under the name “pmserver.cer” to the Keystores folder.

Step 5. In the Windows Explorer window, double-click on the certificate pmserver.cer. In the certificate manager, select the tab “Details” and click the “Copy to File…” button. This launches the Certificate Export Wizard. Click the “Next” button, select the radio button “Cryptographic Message Syntax Standard – PKCS #7 Certificates (.P7B)”, check the checkbox “Include all certificates in the certification path if possible”, and click the “Next” button. Export the certificates under the name pmserver.p7b to folder Keystores. Click “Next”, then “Finish”.

Step 6. In the MS-DOS command window, run the following command:

>keytool -import -alias pmserver -file pmserver.p7b -keystore serverKeystore

Enter keystore password: aaaaaa

Top-level certificate in reply:

Owner: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS=

serban.gavrila@nist.gov

Issuer: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS

=serban.gavrila@nist.gov

Serial number: 186ae6f14015eb86435db3746312cab9

Valid from: Fri Aug 26 12:26:08 EDT 2005 until: Mon Aug 25 12:34:37 EDT 2008

Certificate fingerprints:

MD5: 68:F5:23:C4:7F:11:AF:D5:B1:1D:EB:0A:7B:22:6E:96

SHA1: D6:E0:29:DF:3E:D4:A7:44:F2:86:3E:F9:09:1C:E4:E6:11:93:CE:59

... is not trusted. Install reply anyway? [no]: yes

Certificate reply was installed in keystore

Step 7. Using the same certificate manager, save the CA root certificate base 64-encoded as root.cer, then run the command:

>keytool -import -alias root -file root.cer -keystore serverTruststore

Enter keystore password: aaaaaa

Owner: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS=

serban.gavrila@nist.gov

Issuer: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS

=serban.gavrila@nist.gov

Serial number: 186ae6f14015eb86435db3746312cab9

Valid from: Fri Aug 26 12:26:08 EDT 2005 until: Mon Aug 25 12:34:37 EDT 2008

Certificate fingerprints:

MD5: 68:F5:23:C4:7F:11:AF:D5:B1:1D:EB:0A:7B:22:6E:96

SHA1: D6:E0:29:DF:3E:D4:A7:44:F2:86:3E:F9:09:1C:E4:E6:11:93:CE:59

Trust this certificate? [no]: yes

Certificate was added to keystore

## 2.5 Running the Policy Machine Server

Login as Administrator on the computer that will host the PM engine (server) and run the following command in a MS-DOS command window:

java -cp E:\PolicyMachine\PMServer\server.jar

-Djavax.net.ssl.keyStore=E:\PolicyMachine\Keystores\serverKeystore

-Djavax.net.ssl.keyStorePassword=aaaaaa

-Djavax.net.ssl.trustStore=E:\PolicyMachine\Keystores\serverTruststore

pmserver.PmEngine

Note that you need to update the command to reflect the locations of the server.jar, the server key store and the server trust store. You also need to update the password of your key store.

# 3 Installing the Policy Machine Administrative Tool

## 3.1 Overview

Any PM-authenticated user may use the PM Administrative Tool to manage the Policy Machine data. In addition to being a PM user, the user must also possess a X.509 certificate whose CN value must be the (logon) name of the PM user. Furthermore, the PM engine (server) must trust the CA that issued the user certificate. In the simple case where the same CA issued all certificates (including the PM server’s certificate), this requirement can be satisfied by having the root CA certificate included in the trust stores of the clients (e.g., the Administrative Tool) and the server.

To install the PM Administrative Tool, follow these steps:

Step 1. Log on as Administrator on the computer that will host the PM Administrative Tool.

Step 2. Unpack the distribution file in a folder of your choice. Copy the dot.exe file to a folder included in the path of the user that will run the tool.

Step 3. Obtain a X.509 certificate for the “super” user and store its private key and the certificate into a key store. Store the root certificate(s) in a trust store. Section 3.2 describes a procedure that uses a MS Stand-Alone Certification Authority (CA), which can be found on the Windows 2000 or 2003 Servers if you installed MS Certification Services. Of course, the certificate can be obtained from other CAs.

## 3.2 Obtaining User Certificates

Obtaining a user’s certificate is very similar to obtaining the server’s certificate. Here we detail the procedure for the user “super”. For other users, simply replace “super” with the user’s name everywhere.

Step 1. Log as Administrator on the computer whose CA was used to issue the PM server’s certificate.

Step 2. Create a folder named Keystores. This folder will contain the PM user key store and trust store. The key store is a repository for the user’s private key and certificate. The trust store is a repository for public key certificates that the user trusts. For simplicity, we assume that the same CA issues all certificates. Hence, the trust store needs only contain the CA’s root certificate.

Step 3. Open a MS-DOS command window, cd to the Keystores folder, and run the following commands:

>keytool -genkey -keyalg RSA -alias super -keystore superKeystore

Enter keystore password: aaaaaa

What is your first and last name?

[Unknown]: super

What is the name of your organizational unit?

[Unknown]: CSD

What is the name of your organization?

[Unknown]: NIST

What is the name of your City or Locality?

[Unknown]: Gaithersburg

What is the name of your State or Province?

[Unknown]: Maryland

What is the two-letter country code for this unit?

[Unknown]: US

Is CN=super, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US correct?

[no]: yes

Enter key password for <super>

(RETURN if same as keystore password):

>keytool -certreq -alias super -file super.csr -keystore superKeystore

Enter keystore password: aaaaaa

The file super.csr contains a standard certificate request.

Step 4. Use the MS Internet Explorer to submit the certificate request to the CA installed on the MS Windows Server 2000 or 2003. Namely, open the URL

<http://mymachine/certsrv>

where “mymachine” must be replaced by the server’s host name. From the web page displayed, select the radio button “Request a certificate” and click the “Next” button. From the next page, select the radio button “Advanced request” and click the “Next” button. From the next page, select the radio button “Submit a certificate request using a base64 encoded PKCS #10 file or a renewal request using a base64 encoded PKCS #7 file” and click the “Next” button. Copy the contents of the super.csr file and paste it in the text area titled “Saved Request”. Click the “Submit” button.

In the next page, select the radio button “Base 64 encoded” and select the link “Download CA certificate”. When asked, save the downloaded certificate under the name “super.cer” to the Keystores folder.

Step 5. In the Windows Explorer window, double-click on the certificate super.cer. In the certificate manager, select the tab “Details” and click the “Copy to File…” button. This launches the Certificate Export Wizard. Click the “Next” button, select the radio button “Cryptographic Message Syntax Standard – PKCS #7 Certificates (.P7B)”, check the checkbox “Include all certificates in the certification path if possible”, and click the “Next” button. Export the certificates under the name super.p7b to folder Keystores. Click “Next”, then “Finish”.

Step 6. In the MS-DOS command window, run the following command:

>keytool -import -alias super -file super.p7b -keystore superKeystore

Enter keystore password: aaaaaa

Top-level certificate in reply:

Owner: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS=serban.gavrila@nist.gov

Issuer: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS

=serban.gavrila@nist.gov

Serial number: ...

... is not trusted. Install reply anyway? [no]: yes

Certificate reply was installed in keystore

Step 7. Using the same certificate manager, save the CA root certificate base 64-encoded as root.cer, then run the command:

>keytool -import -alias root -file root.cer -keystore superTruststore

Enter keystore password: aaaaaa

Owner: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS=

serban.gavrila@nist.gov

Issuer: CN=PMCA, OU=CSD, O=NIST, L=Gaithersburg, ST=Maryland, C=US, EMAILADDRESS

=serban.gavrila@nist.gov

Serial number: 186ae6f14015eb86435db3746312cab9

Valid from: Fri Aug 26 12:26:08 EDT 2005 until: Mon Aug 25 12:34:37 EDT 2008

Certificate fingerprints:

MD5: 68:F5:23:C4:7F:11:AF:D5:B1:1D:EB:0A:7B:22:6E:96

SHA1: D6:E0:29:DF:3E:D4:A7:44:F2:86:3E:F9:09:1C:E4:E6:11:93:CE:59

Trust this certificate? [no]: yes

Certificate was added to keystore

## 3.3 Running the Policy Machine Administrative Tool

Step 1. Make sure that the PM server is running on its host computer. Let’s assume that the server’s host computer has the name “musial”.

Step 2. Log as any user on the computer on which you want to run the Admin Tool and run the following command in a MS-DOS command window:

java –cp E:\PolicyMachine\PMAdmin\admin.jar

-Djavax.net.ssl.keyStore=E:\PolicyMachine\Keystores\superKeystore

-Djavax.net.ssl.keyStorePassword=aaaaaa

-Djavax.net.ssl.trustStore=E:\PolicyMachine\Keystores\superTruststore

pmadmin.PmAdmin -enginehost musial -engineport 8080 –debug

Note that you need to update the command with your correct locations and names of the admin.jar, super key store and super trust store, as well as your key store password.

Step 3. The Administrative Tool will request your user name and password. The user name is “super”. The initial password is “super”. At the first log in as super, you should the password by using the “Change Password…” menu.

Now you can use the Admin Tool to create other PM users, grant them the right to perform administrative operations, create certificate for them, etc.

# 4 Installing the User Environment Simulator

## 4.1 Overview

The User Environment Simulator (UES) provides the user with an environment for displaying the user’s virtual object system and executing various PM applications. UES must be run on every target system controlled by the PM engine.

UES contains a kernel simulator and a session manager. Using the session manager GUI, a PM user can log on and create a session. Multiple sessions of the same or different users can exist in the same time on the same computer.

To install the PM Administrative Tool, follow these steps:

Step 1. Log on as Administrator on the computer that will host the UES.

Step 2. Unpack the distribution file in a folder of your choice. Copy the dot.exe file to a folder included in the Administrator’s path.

Step 3. Obtain a X.509 certificate for the computer that will host the UES and store its private key and the certificate into a key store. Store the root certificate(s) in a trust store. The procedure is very similar to those described in Sections 2.4 and 3.2 and will not be further described.

## 4.2 Running the User Environment Simulator

Step 1. Make sure that the PM server is running on its host computer. Let’s assume that the server’s host computer has the name “musial”.

Step 2. Log as Administrator on the target system where you want to run UES. Copy the files simulator.jar, admin.jar, editor.jar from the distribution zip file to folders of your choice. Assume that the target system name is “xarax”.

Step 3. Make sure that the target system has a key store containing its private key and certificate, and a trust store containing the root certificate. Assume that the key store is named “xaraxKeystore” and the trust store is named “xaraxTruststore”.

Step 4. For each PM user that will log on UES on that target host, create a key store containing that user’s private key and certificate, and a trust store containing the root certificate(s) of the CA. The user will need the certificates in order to run the available PM applications.

Step 5. Run the following command in a MS-DOS command window:

java –cp E:\PolicyMachine\PMAdmin\simulator.jar

-Djavax.net.ssl.keyStore=E:\PolicyMachine\Keystores\xaraxKeystore

-Djavax.net.ssl.keyStorePassword=aaaaaa

-Djavax.net.ssl.trustStore=E:\PolicyMachine\Keystores\xaraxTruststore

pmuser.KernelSimulator -enginehost musial -engineport 8080

-simport 8081 -debug

Note that you need to update the command with your correct locations and names of the simulator.jar, xarax’s key store and xarax’s trust store, as well as your key store password.

Step 6. First time UES runs on a target host, you must select the session manager’s “Manager/Configure…” menu and set up the locations of the two PM applications, the Administrative Tool (admin.jar) and the RTF editor (editor.jar).

Step 7. Let users log on the UES by using the login dialog of the session manager. First time a user logs on UES, he/she must select the session’s “Tools/Configure…” menu and set up the locations of his/her key store and trust store, which are needed in order to run PM applications.

# 5 Installing the Applications

There are three applications shipped together with the Policy Machine: a RTF editor (editor.jar), a RTF editor that contains a Trojan horse (thEditor.jar), and a simple workflow application (wkf.jar). They can be run from a user’s session by using the menus, or by opening an object of corresponding type.

To install the PM applications, follow these steps:

Step 1. Log on as Administrator on the computer that will host the UES.

Step 2. Unpack the distribution file in a folder of your choice. Copy the editor.jar, thEditor.jar and wkf.jar to the desired folders.

Step 3. Launch the user environment simulator and activate the “Manager/Configure…” menu. Enter the complete paths for the jar files of the admin tool, editor or Trojan horse editor, and workflow application.

# Appendix A. Active Directory Classes and Attributes

## A.1 The User Class

The LDAP name of the class is *pmClassUser*. An object of this class represents a user as defined in in the formal specification of the Policy Machine. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): an enterprise-unique user identifier.

**pmName** (mandatory, alphanumeric, single-valued): the human-friendly name of the PM user.

**pmFullName** (mandatory, alphanumeric, single-valued): user’s full name.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmPassword** (mandatory, alphanumeric, single-valued): the hashed user password.

**pmToAttr** (optional, alphanumeric, multi-valued): the id of an attribute this user is assigned to.

**pmToConnector** (optional, alphanumeric, single-valued): the id of the connector node, if the user is assigned to it. This happens only when the user is not assigned to a user attribute.

## A.2 The User Attribute Class

The LDAP name of the class is pmClassUserAttribute. An object of this class represents a user attribute as defined in [1]. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): an enterprise-unique attribute identifier.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly, unique, attribute identifier.

**pmDescription** (mandatory, alphanumeric, single-valued): attribute description.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmFromUser** (optional, alphanumeric, multi-valued): the id of a user assigned to this attribute.

**pmToAttr** (optional, alphanumeric, multi-valued): the id of an attribute this attribute is assigned to.

**pmFromAttr** (optional, alphanumeric, multi-valued): the id of an attribute assigned to this attribute.

**pmToPolicy** (optional, alphanumeric, multi-valued): the id of a policy this attribute is assigned to (i.e., this → policy id).

**pmToOpSet** (optional, alphanumeric, multi-valued): the id of an operation set this attribute is assigned to. The operation set contains operations allowed to users possessing this attribute.

**pmToConnector** (optional, alphanumeric, single-valued): the id of the connector node, if this attribute is assigned to the connector node. This happens only when the user attribute is not assigned to another user attribute or to a policy class.

**pmProperty** (optional, alphanumeric, multi-valued): a property of this attribute, usually in the format “<property\_name>=<property\_value>”. This attribute can be used in EVER functions for attribute selection.

## A.3 The Policy Class Class

The LDAP name of the class is *pmClassPolicy*. An object of this class represents a policy class as defined in [1]. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): an enterprise-unique policy identifier.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly, unique, policy identifier.

**pmDescription** (mandatory, alphanumeric, single-valued): policy description.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmToConnector** (mandatory, alphanumeric, single-valued): the id of the connector node. Note that a policy class is always assigned to the connector node.

**pmFromUserAttr** (optional, alphanumeric, multi-valued): the id of a user attribute assigned to this policy.

**pmFromObjAttr** (optional, alphanumeric, multi-valued): the id of an object attribute assigned to this policy.

**pmProperty** (optional, alphanumeric, multi-valued): a property of this policy, usually in the format “<property\_name>=<property\_value>”. This attribute can be used in EVER functions for policy selection.

## A.4 The Object Class Class

The LDAP name of the class is *pmClassObjectClass*. The PM objects as defined in [1] may be of different classes (e.g., files, or directories, or ports, etc.) An object of the class *pmClassObjectClass* represents a class of objects. An object class mainly defines the set of operations valid on objects of that class. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): an enterprise-unique object class identifier.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly, unique, object class name (e.g., “file”, “directory”, “printer”).

**pmDescription** (mandatory, alphanumeric, single-valued): the object class description.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmOp** (optional, alphanumeric, multi-valued): the name of an operation valid for objects of this class (e.g., “File read” for reading objects of class “File”).

## A.5 The Operation Set Class

The LDAP name of the class is *pmClassOperationSet*. The PM assigns attributes to operation sets. An object of this class represents an operation set, in which all operations belong to the object class. An operation set has a name, some member operations, and possible pointers to and from assigned attributes. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): a unique identifier for the operation set.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly identifier for the operation set.

**pmDescription** (mandatory, alphanumeric, single-valued): operation set description.

**pmObjClass** (mandatory, alphanumeric, single-valued): no longer used.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmOp** (optional, alphanumeric, multi-valued): the name of an operation member of this set.

**pmFromAttr** (optional, alphanumeric, multi-valued): the id of a user attribute assigned to this operation set.

**pmToAttr** (optional, alphanumeric, multi-valued): the id of an object attribute this operation set is assigned to.

**pmToConnector** (optional, alphanumeric, single-valued): the id of the connector object if this operation set is assigned to the connector node. This happens only when the operation set is not assigned to an object attribute.

## A.6 The Virtual Object Class

The LDAP name of the class is *pmClassVirtualObject*. A virtual object can be used to designate:

* An actual file. In this case, the virtual object has the PM class File and should specify the computer host and the path of the actual file.
* A subset of the PM data that corresponds to a PM entity and possibly to all its ascendants with respect to the assignment relation. In this case, we say that the virtual object represents the sub-graph consisting of the PM entity and possibly its ascendants. The virtual object has the PM class User, User attribute, Object, Object attribute, Policy class, Connector, or Operation set, according to the type of the PM entity. The virtual object should specify the id and name of the PM entity, and whether the virtual object represents only the PM entity or it also includes the ascendant entities.

Please note that the PM class of a virtual object is entirely different from the AD class of that object.

Whenever a virtual object is created, an object attribute is also automatically created and inserted into the object attributes hierarchy instead of the object itself. We say that this object attribute and the object are *associated* to each other.

The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): an enterprise-unique object identifier.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly, unique, object name.

**pmDescription** (mandatory, alphanumeric, single-valued): object description.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmAssocAttr** (mandatory, alphanumeric, single-valued): the id of the object attribute associated to this object.

**pmObjClass** (mandatory, alphanumeric, single-valued): the name of the PM class of this object. There are a few predefined PM classes, among them File, User, User attribute, Object, Object attribute, Policy class, Connector, Operation set. The administrator can define custom classes.

**pmHost** (optional, alphanumeric, single-valued): the name or IP address of the target system where the content of an object of class File actually resides. This attribute is mandatory for objects of class File.

**pmPath** (optional, alphanumeric, single-valued): the path name of the content of an object of class File. This attribute is mandatory for objects of class File.

**pmOriginalId** (optional, alphanumeric, single-valued): the id (the value of the attribute pmId) of the PM entity represented by this object. This attribute is mandatory for objects of class User, User attribute, Object, Object attribute, Policy class, Connector, Operation set.

**pmOriginalName** (optional, alphanumeric, single-valued): the name (the value of the attribute pmName) of the PM entity represented by this object. This attribute is mandatory for objects of class User, User attribute, Object, Object attribute, Policy class, Connector, Operation set.

**pmIncludesAscendants** (optional, Boolean, single-valued): specifies whether this object represents only the PM entity identified by pmOriginalId and pmOriginalName (when its value is FALSE), or the PM entity together with all its ascendants in the general PM graph (when its value is TRUE). This attribute is mandatory for objects of class User, User attribute, Object, Object attribute, Policy class, Connector, Operation set.

## A.7 The Object Attribute Class

The LDAP name of the class is *pmClassObjectAttribute*. An object of this class represents a PM object attribute as described in [1]. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): an enterprise-unique attribute identifier.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly, unique attribute identifier. For the object attribute associated to an object, this field has the value of the pmName field of the associated object.

**pmDescription** (mandatory, alphanumeric, single-valued): attribute description.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmAssocObj** (optional, alphanumeric, single-valued): if this attribute is present, this object attribute is associated to an object with the same name as the value of pmName, and the value of this attribute is the id of that object.

**pmToAttr** (optional, alphanumeric, multi-valued): the id of an object attribute this object attribute is assigned to.

**pmFromAttr** (optional, alphanumeric, multi-valued): the id of an object attribute assigned to this object attribute.

**pmToPolicy** (optional, alphanumeric, multi-valued): the id of a policy this attribute is assigned to.

**pmFromOpSet** (optional, alphanumeric, multi-valued): the id of an operation set assigned to this object attribute.

**pmToConnector** (optional, alphanumeric, single-valued): the id of the connector node, if this object attribute is assigned to the connector node. This happens only when this object attribute is not assigned to another object attribute or to a policy class.

**pmProperty** (optional, alphanumeric, multi-valued): a property of this object attribute, usually in the format “<property\_name>=<property\_value>”. This attribute can be used in EVER functions for object attribute selection.

## A.8 The Session Class

The LDAP name of the class is *pmClassSession*. An object of this class represents a user session. The Session Manager opens a session on behalf of a user, on a target system controlled by PM, and with an empty set of active attributes. The PM engine may augment the session’s set of active attributes when processing user’s permission requests. Note that an active attribute remains active for the duration of the session.

This class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): a unique session identifier.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly unique session identifier.

**pmHost** (mandatory, alphanumeric, single-valued): the *name* of the host computer where the session was opened.

**pmUserId** (mandatory, alphanumeric, single-valued): the id of the user on behalf of which this session was opened.

**pmTimestamp** (mandatory, date, single-valued): the timestamp of the session start.

**pmAttr** (optional, alphanumeric, multi-valued): the id of an active (in this session) user attribute.

## A.9 The VOS Node Class

When a user logs on a host controlled by the PM, the session manager opens a session with an empty set of active attributes for that user. The session displays the user’s virtual object system (VOS), which consists of the PM objects that user can access, their policy classes, and the connector node. To compute and present the user’s VOS, the PM engine uses objects of the VOS Node class, which represent various PM entities.

The LDAP name of the class is *pmClassVosNode*. This class has the following attributes:

**pmId** (mandatory, alphanumeric, single-valued): a unique id for this node. Note that this id is different from the id of the original PM entity this node represents.

**pmOriginalId** (mandatory, alphanumeric, single-valued): the id of the original PM entity this node represents.

**pmOriginalName** (mandatory, alphanumeric, single-valued): the name of the original PM entity this node represents.

**pmSessId** (mandatory, single-valued): the id of the session for which this VOS node was created.

**pmUserId** (mandatory, alphanumeric, single-valued): the id of the user for which this VOS node was created.

**pmUserName** (mandatory, alphanumeric, single-valued): the name of the user for which this VOS node was created.

**pmObjClass** (mandatory, alphanumeric, single-valued): the AD class of the PM entity this node represents. For example, it is “pmClassConnector” for the connector node, “pmClassPolicy” for policy classes, etc.

**pmIsObj** (mandatory, Boolean, single-valued): whether the node represents an object. In this case, the pmOriginalId attribute contains the id of the object attribute associated with the object, and not the object itself.

**pmAscNode** (optional, alphanumeric, multi-valued): the id of an ascendant node.

**pmDescNode** (optional, alphanumeric, multi-valued): the id of a descendant node.

## A.10 The Admin VOS Node Class

A user’s session may display the user’s administrative virtual object system (AdVOS), which consists of the PM entities that user can access and modify. To compute and present the user’s AdVOS, the PM engine uses objects of the Admin VOS Node class, which represent various PM entities.

The LDAP name of the class is *pmClassAdminVosNode*. This class has the following attributes:

**pmId** (mandatory, alphanumeric, single-valued): a unique id for this node. Note that this id is different from the id of the original PM entity this node represents.

**pmName** (mandatory, alphanumeric, single-valued): the name of this node and of the original PM entity this node represents.

**pmOriginalId** (mandatory, alphanumeric, single-valued): the id of the original PM entity this node represents.

**pmType** (mandatory, alphanumeric, single-valued): the type of the original PM entity this node represents. It has one of the values u (for user), a (user attribute), b (object attribute), o (object attribute associated to an object), p (policy class), c (connector).

**pmSessId** (mandatory, single-valued): the id of the session for which this node was created.

**pmUserId** (mandatory, alphanumeric, single-valued): the id of the user for which this node was created.

**pmUserName** (mandatory, alphanumeric, single-valued): the name of the user for which this node was created.

**pmAscNode** (optional, alphanumeric, multi-valued): the id of an ascendant node.

**pmDescNode** (optional, alphanumeric, multi-valued): the id of a descendant node.

## A.11 The Attribute Set Class

The LDAP name of the class is *pmClassAttributeSet*. An object of this class represents a set of user attributes. Attribute sets are involved in restrictions such as subject attribute constraints defined in [1]. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the id of the attribute set, unique.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly id of the attribute set, unique.

**pmAttr** (optional, alphanumeric, multi-valued): the id of an attribute member of the attribute set.

## A.12 The SAC (Subject Attribute Constraint) Class

The LDAP name of the class is *pmClassSac*. An object of this class represents a subject attribute constraint (SAC) as defined in [1]. The constraint has an id, a name, and a list of attribute sets that must be pairwise-disjoint. An attribute set *as* satisfies the constraint if and only if *as* has no common elements with more than one set of the constraint. In PM, the set of active attributes of any session must satisfy all SAC constraints for the session duration.

The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the id of the constraint, unique.

**pmName** (mandatory, alphanumeric, single-valued): the human-friendly id of the constraint, unique.

**pmDescription** (mandatory, alphanumeric, single-valued): description of the constraint.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information.

**pmAttrSet** (optional, alphanumeric, multi-valued): the id of an attribute set.

## A.13 The Connector Class

The LDAP name of the class is *pmClassConnector*. The Policy Machine engine uses only one object of this class to connect all the nodes in the PM graph (the graph is defined in [1]). We call this object the connector node. The PM engine creates automatically this object when it starts up on an empty PM database. Its id is 1 and its name is “PM”.

Any policy class that is created will be assigned to the connector node and will remain assigned to it until it is deleted.

Any new user who is not assigned to an existing user attribute will be assigned to the connector node. After its assignment to a user attribute, that assignment will be deleted. When all assignments of a user to attributes are deleted, that user is assigned to the connector node.

Any new user attribute that is assigned at creation time to a user (but not to a policy class or to another attribute) will be initially assigned to the connector node. When that user attribute is later assigned to another attribute or to a policy class, that assignment is deleted. When all assignments of a user attribute to other user attributes or policy classes are deleted, that user attribute is assigned to the connector node.

Similar things happen for object attributes and operation sets.

The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the id of the connector object.

**pmName** (mandatory, alphanumeric, single-valued): the name of the connector object (“PM”).

**pmFromUser** (optional, alphanumeric, multi-valued): the id of a user that is not assigned to a user attribute.

**pmFromUserAttr** (optional, alphanumeric, multi-valued): the id of a user attribute that is not assigned to another user attribute or policy class.

**pmFromObjAttr** (optional, alphanumeric, multi-valued): the id of an object attribute that is not assigned to another object attribute or policy class.

**pmFromPolicy** (optional, alphanumeric, multi-valued): the id of a policy class.

**pmFromOpSet** (optional, alphanumeric, multi-valued): the id of an operation set that is not assigned to a user attribute or to an object attribute.

## A.14 The Event Class

This class describes administrative events, like the creation of a new session. Its LDAP name is *pmClassEvent*. The events are generated by the engine and inserted in a queue. Other components of the PM (for example, the Admin Tool) update their GUI by consuming the events. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the event.

**pmEvent** (mandatory, alphanumeric, single-valued): a name that identifies the type of the event, e.g., “created”, “deleted”, etc.

**pmOriginalId** (mandatory, alphanumeric, single-valued): the id of the event’s object, e.g., the id of the newly created session.

**pmObjClass** (mandatory, alphanumeric, single-valued): a name that identifies the PM class of the event’s object, e.g., “pmClassSession” for a session object.

**pmTimestamp** (mandatory, date, single-valued): timestamp of the event.

## A.15 The Host Class

This class describes target systems controlled by the PM. Its LDAP name is *pmClassHost*. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the host object.

**pmName** (mandatory, alphanumeric, single-valued): a name that identifies the host (might be its IP address).

**pmDescription** (mandatory, alphanumeric, single-valued): a short description of the host.

**pmIsDomController** (mandatory, Boolean, single-valued): specifies whether the host is a domain controller.

**pmOtherInfo** (optional, alphanumeric, single-valued): other information about the host.

**pmHttpPort** (optional, alphanumeric, single-valued): if present, specifies the port used by a web server installed on the host.

**pmCgiName** (optional, alphanumeric, single-valued): if present, specifies the name of a Common Gateway Interface (CGI) script installed on this host and serving as administrative agent.

**pmVirtualDirectory** (optional, alphanumeric, single-valued): if present, specifies the virtual directory where the administrative agent is installed.

**pmIpAddress** (optional, alphanumeric, single-valued): the (static) IP address of this host.

## A.16 The Deny Class

The Deny class describes deny constraints. Its LDAP name is *pmClassDeny*. Its PM attributes are:

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the constraint object.

**pmName** (mandatory, alphanumeric, single-valued): a name that identifies the constraint.

**pmType** (mandatory, alphanumeric, single-valued): specifies the type of the deny constraint, user id-based, intra-session active attribute-based, or inter-sessions active attribute based.

**pmOriginalId** (mandatory, alphanumeric, single-valued): the user id or user attribute id, based on the type of constraint.

**pmIsIntersection** (mandatory, Boolean, single-valued): a Boolean specifying whether the deny constraint is to use the intersection or the union of the object containers.

**pmOp** (optional, alphanumeric, multi-valued): specifies a denied operation.

**pmAttr** (optional, alphanumeric, multi-valued): specifies the id of an object container (attribute), possibly prefixed with the ‘!’ sign to indicate the complement of that container.

## A.17 The Task Class

The LDAP name of the class is *pmClassTask*. An object of this class represents a (business) task, i.e., a set of capabilities needed to perform the task. Tasks are involved in static constraints as defined in [1]. The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the task id, unique.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly id of the task, unique.

**pmCap** (optional, alphanumeric, multi-valued): a string representing a capability within a policy class, i.e., a triple <operation name>,<object id>,<policy class id>, with the elements separated by commas.

## A.18 The Scon Class (static constraints on user capabilities)

The LDAP name of the class is *pmClassStaticConstraint*. An object of this class represents a static constraint <*k*, (*t*1, …, *tn*)>, where *k* is a small integer called threshold, and *t*1, …, *tn* are tasks. Its meaning is that *no* user may be authorized for *more* than *k* tasks from *t*1, …, *tn*.

The class has the following PM-related attributes:

**pmId** (mandatory, alphanumeric, single-valued): the id of the constraint, unique.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly id of the constraint, unique.

**pmThreshold** (mandatory, integer, single-valued): the value of the threshold *k*.

**pmTask** (optional, alphanumeric, multi-valued): a task id that identifies an object of class pmClassTask that represents one of the tasks *t*1, …, *tn*.

## A.19 The Scona Class (static constraints on attributes)

The LDAP name of the class is *pmClassScona*. An object of this class represents a static constraint <*ua*, *opset*, *oa*>, where *ua* is a user attribute, *opset* is an operation set, and *oa* is an object attribute. It represents the following constraint: the user attribute *ua* cannot be authorized for any operation in *opset* on the object attribute *oa*.

**pmId** (mandatory, alphanumeric, single-valued): the id of the constraint, unique.

**pmName** (mandatory, alphanumeric, single-valued): a human-friendly id of the constraint, unique.

**pmPc** (mandatory, alphanumeric, multi-valued): the policy class id.

**pmUserAttr** (mandatory, alphanumeric, single-valued): the user attribute id.

**pmObjAttr** (mandatory, alphanumeric, single-valued): the object attribute id.

**pmOp** (optional, alphanumeric, multi-valued): an operation in the operation set.

## A.20 The Script class

The objects of this class represent scripts in the EVER language compiled and stored in the active directory. The LDAP name of this class is pmClassScript.

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the EVER script.

**pmName** (mandatory, alphanumeric, single-valued): the name of the script, as declared in the “script *script\_name*” clause.

**pmIsEnabled** (mandatory, Boolean, single-valued): whether the script is enabled (to be used for event processing) or not.

**pmCount** (optional, integer, single-valued): the number of rules in this script.

**pmFirst** (optional, alphanumeric, single-valued): the id of the script’s first rule. If the script does not have rules, this attribute must not be set.

**pmLast** (optional, alphanumeric, single-valued): the id of the script’s last rule. If the script does not have rules, this attribute must not be set.

## A.21 The Rule class

An object of this class represents an event/response relation (or rule) compiled to intermediary code. The LDAP name of this class is pmClassRule*.*

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the event-response rule.

**pmLabel** (optional, alphanumeric, single-valued): an optional label of the event-response rule.

**pmNext** (optional, alphanumeric, single-valued): the unique id of the next rule in the script. If this is the last rule in the script, this attribute must not be set.

**pmPrev** (optional, alphanumeric, single-valued): the unique id of the previous rule in the script. If this is the first rule in the script, this attribute must not be set.

**pmRank** (optional, integer, single-valued): the rank of this rule in the script, starting with 0.

**pmCount** (optional, integer, single-valued): the number of actions in this rule.

**pmFirst** (optional, alphanumeric, single-valued): the id of the rule’s first action. If the rule does not have actions, this attribute must not be set.

**pmLast** (optional, alphanumeric, single-valued): the id of the rule’s last action. If the rule does not have actions, this attribute must not be set.

**pmEventPattern** (mandatory, alphanumeric, single-valued): the unique id of the event pattern this event-response rule applies to.

## A.22 The EventPattern class

An object of this class represents an event pattern. The engine tries to match each event received from other components of the PM system against a list of event patterns. When the event matches a pattern, the engine applies the administrative commands (or actions) associated with the pattern to modify the PM state. The LDAP name of this class is *pmClassEventPattern*.

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the event pattern.

**pmUserSpec** (optional, alphanumeric, multi-valued): the type and id of a user or user attribute. The format is “\*” or <type>|<id> or <type>:<id>, where type is “u” for users, and “a” for user attributes.

**pmIsActive** (mandatory, Boolean, single-valued): specifies whether the user of the event’s subject must be active (true) in or simply a member (false) of the policy classes specified in this event pattern, in order for the event to match the pattern.

**pmIsAny** (mandatory, Boolean, single-valued): specifies whether the user of the event’s subject must be (active as) a member of any (true) or each (false) policy class specified in this event pattern, in order for the event to match the pattern.

**pmPolicySpec** (optional, alphanumeric, multi-valued): the type and id of a policy class. The format is “\*” or <type>|<id> or <type>:<id>, where type is “p” for policy class.

**pmOpSpec** (optional, alphanumeric, multi-valued): the name of an event, like “Create object” or “Delete session”. Its value must be the value of the pmName attribute of an object of class pmClassEventName (i.e., it must be an event name), or the wildcard “\*”.

**pmObjSpec** (optional, alphanumeric, multi-valued): the type and id of an object, object attribute, or object class. The format is “\*” or <type>|<id> or <type>:<id>, where type is “ob” for object, “b” for object attribute, and “oc” for class.

## A.23 The Action class

An object of this class represents an (administrative) command (or action) of an event/response rule. The object is used in the intermediary code of a EVER script.

The LDAP name of this class is *pmClassAction*.

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the action.

**pmType** (mandatory, alphanumeric, single-valued): the action type (e.g., “grant”, “assign”, “assign like”).

**pmIsIntrasession** (mandatory, boolean, single-valued): Applies only to Deny actions based on an active attribute. If true, the deny constraint applies to attributes active in the current session. Otherwise, the deny constraint applies to attributes active in any session of the current user.

**pmIsIntersection** (mandatory, boolean, single-valued): Applies only to Deny actions. If true, the deny constraint applies to the intersection of the specified object containers, otherwise to their union.

**pmNext** (optional, alphanumeric, single-valued): the unique id of the next action in the rule. If this is the last action of the rule, this attribute must not be set.

**pmPrev** (optional, alphanumeric, single-valued): the unique id of the previous action in the rule. If this is the first action in the rule, this attribute must not be set.

**pmRank** (optional, integer, single-valued): the rank of this action in the rule, starting with 0.

**pmOpnd1** (optional, alphanumeric, multi-valued): the id of this action’s first operand.

**pmOpnd2** (optional, alphanumeric, multi-valued): the id of this action’s second operand.

**pmOpnd3** (optional, alphanumeric, multi-valued): the id of this action’s third operand.

**pmOpnd4** (optional, alphanumeric, multi-valued): the id of this action’s fourth operand.

## A.24 The Operand class

An object of this class represents an operand of an administrative command of an event/response relation, compiled to intermediary code. An operand in general is a PM entity of type user, user attribute, object, object attribute, policy class, base, operation, operation set, or object class. The operand can represent the return value of a function, in which case the value is an array of PM entities.

The LDAP name of this class is *pmClassOperand*.

**pmId** (mandatory, alphanumeric, single-valued): the unique id of the operand.

**pmType** (mandatory, alphanumeric, single-valued): the operand type. The type is “u” for user, “a” for user attribute, “ob” for object, “b” for object attribute, “p” for policy class, “c” for connector, “op” for operation, “os” for operation set, “oc” for class. **pmIsFunction** (mandatory, Boolean, single-valued): indicates whether the operand represents the return value of a function.

**pmIsSubgraph** (mandatory, Boolean, single-valued): indicates whether the operand represents a subgraph of the PM graph.

**pmIsComplement** (mandatory, Boolean, single-valued): indicates whether the operand represents the complement of a container.

**pmOriginalId** (optional, alphanumeric, single-valued): the operand’s original id. For a PM entity like user or user attribute, etc., this is the id of the PM entity. For a function call, this is the id of an object of AD class pmFunctionCall – see below. For an operation, this attribute should be omitted – operations do not have ids.

**pmOriginalName** (mandatory, alphanumeric, single-valued): the operand’s original name. For a PM entity like user, user attribute, etc., this is the name of the PM entity. For a function call, this is the name of the function.

**pmArgs** (optional, alphanumeric, single-valued): If this operand represents the return value of a function, then this attribute contains the ids of the function’s arguments in the order in which they appear in the function call, separated by ‘|’. An argument is an object of pmActionOperand class.