www.internet2.edu

DCN Software Suite v0.5.1: OSCARS Policy Service User Guide

Table of Contents

1 Overview

1.1 About this Document

This document will guide you through the installation and use of the OSCARS Policy Service. The OSCARS Policy Service allows an administrator to define rules for classes of users that provision dynamic circuit networks. It allows rules such as those that define limits to be placed on the bandwidth of a single circuit or control access to individual network elements. This document assumes that you have the OSCARS IDC software installed. This document also assumes basic familiarity with the UNIX command-line. It does NOT assume any previous familiarity with Java or XML.

1.2 Hardware and Software Requirements

1.2.1 System Requirements

The OSCARS Policy Service requires at minimum a single PC that will act as a web server for processing requests. Most modern PCs should be suitable for running the software. The following specifications are the minimum requirements for most installations:

- 1Ghz Processor (preferably X86 architecture), 1GB memory
- Linux/Unix Operating System
- Basic Internet connectivity
- OSCARS AA Service

Requirements may be greater for systems running the OSCARS Policy Service on the same machine as multiple other components of the DCN Software Suite.

1.2.2 Firewall Requirements

The OSCARS Policy Service runs on TCP port 8080 and port 8443 by default.

1.2.3 Third-Party Library and Package Requirements

Installing and running the OSCARS IDC requires the following software packages:

Name	Supported Version	Download Location
MySQL	5.0+	http://dev.mysql.com/downloads/mysql/

Java Development Kit (JDK)	5.0	http://java.sun.com/javase/downloads/index_jdk5.jsp
Tomcat	5.5	http://tomcat.apache.org/download-55.cgi
Axis2	1.4.1	http://ws.apache.org/axis2/download/1_4_1/download.cgi
Ant	1.7	http://ant.apache.org/bindownload.cgi

1.3 Downloading the Policy Service software

The Policy Service software is part of the DCN Software Suite. It can be downloaded at:

- https://wiki.internet2.edu/confluence/display/DCNSS

After downloading the DCN software suite, you may unpack it with the following commands:

```
% gunzip dcn-software-suite-0.5.X.tar.gz
% tar -xvf dcn-software-suite-0.5.X.tar
```

This will create a directory called `dcn-software-suite-0.5.X`. The OSCARS Policy Service is located in the subdirectory `dcn-software-suite-0.5.X/policy`.

2 Preparing your Environment

This section details how to install and configure perquisite software on the machine that will be running the IDC. The following prerequisite steps are detailed:

1. Install MySQL
2. Install the Java Development Kit and set the JAVA_HOME environment variable
3. Install the Tomcat web server and set the CATALINA_HOME environment variable
4. Install Ant and add it to your PATH environment variable

In addition the Axis2 libraries from Apache must be installed. This will be done automatically by the OSCARS installation script as described in Section 3.

2.1 MySQL

MySQL is the database used to maintain policy information. You may install MySQL in one of two ways: manually, by installing a package downloaded from the MySQL web site OR automatically, using your operating system's package manager:

2.1.1 Install Option 1: Manual Installation

Download the MySQL package from the MySQL web site at:

- http://dev.mysql.com/downloads/mysql

Installing MySQL in this manner is beyond the scope of this document but installation instructions may be found at:

- http://dev.mysql.com/doc

2.1.2 Install Option 2: Automatic Installation with a Package Manager

Download and install MySQL through a package manager if your operating system runs such a service. A few common package managers are up2date (RedHat), apt-get (Debian), and yum. You may install MySQL using a command such as:

```
% up2date mysql-server
```

Consult specific package managers for the exact command and package name.

2.2 Java Development Kit (JDK)

Java is the programming language in which the OSCARS Policy Service software was created and provides the environment in which it runs. In addition to running the software, the Java Development Kit (JDK) also contains utilities required for compiling the source code. This section details installation and configuration related to this package.

2.2.1 Do I already have the right version of Java?

Many systems come pre-installed with Java. To install the Policy Service, your system must not only have Java Runtime Environment (JRE) version 5 but also the various compilers and utilities. To verify that you have the necessary Java environment, issue the following command:

```
% javac -version
```

If the first line of output reads `javac 1.5.0_X`, you should not need to install the Java Development Kit and may skip to section **2.2.3 Setting the JAVA_HOME Environment Variable**. If you get “command not found” or the version number is less than 1.5, you may need to install JDK 5.0 and should proceed to **2.2.2 Download and Installation**.

NOTE: If you are not running the SUN distribution of Java you may encounter issues. The GNU and IBM versions of Java are not fully tested and some users have reported problems. It is recommended you run the SUN distribution of Java.

2.2.2 Download and Installation

You may download JDK 5.0 from Sun's web site at:

- http://java.sun.com/javase/downloads/index_jdk5.jsp

It is recommended that you download the latest update of **JDK Version 5**. Choose the package most suitable for your operating system.

Once downloaded, unpack the file; this should create a new folder named something similar to "jdk1.5.0_X". The final step of installation is to move this folder to an easily accessible place. We recommend renaming the folder to java5 in /usr/local with the following command:

```
% sudo mv jdk1.5.0_X /usr/local/java5
```

The location may be anywhere you choose – just make sure you note the location as it is required for setting the JAVA_HOME environment variable in the next section.

2.2.3 Setting the JAVA_HOME Environment Variable

Once Java is installed, you need to set the JAVA_HOME environment variable with its location. This variable is required by the Tomcat web server (see section **2.3 Tomcat)** to run. To set this environment variable, issue these commands:

```
% JAVA_HOME=/usr/local/java5
% export JAVA_HOME
```

You may permanently set this variable (recommended) by adding the above commands to the profile file in your home directory (i.e. .bash_profile or .profile).

2.2.4 Optional: Adding JAVA_HOME/bin To Your PATH Variable

This step is optional but may make issuing commands easier in later steps. You should add the folder JAVA_HOME/bin to your PATH environment variable so that you can easily access the `keytool` command for issuing certificates. To update your PATH variable, issue these commands:

```
% PATH=$PATH:$JAVA_HOME/bin
% export PATH
```

You may permanently set this variable (recommended) by adding the above commands to the profile file in your home directory (i.e. .bash_profile or .profile).

2.3 Tomcat

Tomcat is a Java-based application container in which the OSCARS Policy Service software runs. This section details installation and basic configuration of Tomcat.

2.3.1 Download and Installation

You may download Tomcat from the project's web site at:

- http://tomcat.apache.org/download-55.cgi

It is recommended you download **Tomcat Version 5.5**.

NOTE: Tomcat 6.0 is NOT currently supported by the software.

Once downloaded, unpack the downloaded file; this should create a new folder named something similar to "apache-tomcat-5.5.X". The final step of installation is to move this folder to an easily accessible place. We recommend renaming the folder to tomcat in /usr/local with the following command:

```
% sudo mv apache-tomcat-5.5.X /usr/local/tomcat
```

The location may be anywhere you choose – just make sure you note the location as it is required for setting the CATALINA_HOME environment variable in the next section.

NOTE: It is NOT RECOMMENDED that you download Tomcat with a package manager such as up2date, yum, or apt-get. Many users have reported difficulty with this method. If you install Tomcat using this method be aware that some of the environment variables and other settings may vary from what is contained within this document.

2.3.2 Setting the CATALINA_HOME Environment Variable

Once Tomcat is installed, you need to set the CATALINA_HOME environment variable with its location. This variable is required by the Tomcat web server to run. To set this environment variable, issue these commands:

```
% CATALINA_HOME=/usr/local/tomcat
% export CATALINA_HOME
```

You may permanently set this variable (recommended) by adding the above commands to the profile file in your home directory (i.e. .bash_profile or .profile).

2.3.3 Starting/Stopping the Tomcat Server

You may start Tomcat with the following command:

```
% $CATALINA_HOME/bin/startup.sh
```

You shutdown the Tomcat server with the following command:

```
% $CATALINA_HOME/bin/shutdown.sh
```

2.3.4 Verifying a Successful Installation

To verify installation was successful, startup the Tomcat server with the following command:

```
% $CATALINA_HOME/bin/startup.sh
```

After starting the server, point a web browser to port 8080 of the machine on which you installed Tomcat with the following URL:

- http://your-machine-name:8080

If installation was successful, a web page will load with the Tomcat logo and a message that reads “If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!”

2.3.5 Configuring SSL

You may configure Tomcat to use SSL so that all requests and responses to the server are encrypted. This is not required but highly recommended. Information on this process can be found at:

- http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html

2.4 Ant

Ant is a tool that is used to build the Policy Service code and deploy various configuration files (think of it as “make” for Java). This section details how to install and configure Ant.

2.4.1 Download and Installation

Download Ant from the project’s web site at:

- http://ant.apache.org/bindownload.cgi

Most IDC testing has been done with **Version 1.7**. Unpack and install Ant with the following commands:

```
% unzip apache-ant-1.7.0-bin.zip
% sudo mv apache-ant-1.7.0 /usr/local/ant
```

You are not required to install the downloaded folder in /usr/local/ant but you should note where it is installed as this information is needed in later steps.

2.4.2 Setting the ANT_HOME Environment Variable

Once Ant is installed, you need to set the ANT_HOME environment variable with Ant’s location. To set this environment variable, issue these commands:

```
% ANT_HOME=/usr/local/ant
% export ANT_HOME
```

You may permanently set this variable (recommended) by adding the above commands to the profile file in your home directory (i.e. .bash_profile or .profile).

2.4.3 Adding ANT_HOME/bin to Your PATH Variable

It is recommended you add the Ant bin directory to your PATH environment variable. This will allow ant command-line tools to be found when you type-in the command name. To set this environment variable, issue these commands:

```
% PATH=$PATH:$ANT_HOME/bin
% export PATH
```

You may permanently set this variable (recommended) by adding the above commands to the profile file in your home directory (i.e. .bash_profile or .profile).

3 Installing the Policy Service Software

This section details how to install the OSCARS Policy Service Software. It assumes you have installed all the prerequisites as described in the previous section (Preparing your Environment). This section will cover installing a Policy Service.

3.1 Installing the Policy Service

Assuming you have downloaded and unpacked the OSCARS policy service, the first step is to change your current working directory to the ‘policy’ directory of the DCN Software Suite Package:

```
% cd dcn-software-suite-0.5.X/policy
```

The next step is to run the `do_build.sh` command to compile the software and build the required database tables:

```
% ./do_build.sh

--- Checking prerequisites...
    We seem to be in the correct directory
    Found ant
    Environment variable CATALINA_HOME is set to
/usr/local/tomcat
    Environment variable DOMAIN_HOME is not set.
Continuing without it.
    Axis2 library not found.
    Rampart library not found.
```

The first set of prompts check if Tomcat and Axis2 are installed. If Axis2 is not installed you will see messages like those shown above. You should answer 'y' to the next two questions and proceed as shown below:

```
- Axis2-1.4.1 with Rampart-SNAPSHOT installation not
detected. Should I build it for you y/n?y
    OK, will build Axis2-1.4.1 for you.

- Axis2-1.4.1 is not deployed. Should I do this for you
y/n? y
    OK, will deploy Axis2 for you.
--- Downloading Axis2...
```

You should see output of Axis2 and the module Rampart downloading. The script will then restart your Tomcat server. The output for the Axis2/Rampart download and the Tomcat restart is not shown in this document, as it is rather lengthy. After restarting Tomcat you should see the following:

```
--- Your kit looks good.
-  Input the hostname for this Policy Service. Leave
   blank for "myhost.example.org":
       Using myhost.example.org.
```

The first prompt asks for the hostname of your machine. If the hostname in quotations is correct then leave blank otherwise enter the correct name before proceeding. Next you will be prompted for information related to MySQL:

```
- Install databases y/n? y
    OK, will install databases.
    Found mysql client at /usr/bin/mysql
- Input the MySQL server hostname. Leave blank for
localhost:
    Using localhost .
- Input a privileged MySQL username on that host. Leave
blank for root:
    Using root .
- Input the password for the privileged account:
    Privileged account access verified.
```

The first set of prompts asks for the credentials of a privileged MySQL user that can create new databases. The default privileged user is 'root' but if this was changed during MySQL setup you may specify the correct info here. The next prompt asks for information about a non-privileged user as shown below:

```
- Input a MySQL username the Policy Service will use to
connect to the databases.
```

```
  -- This name and password must match the
hibernate.connection.username and password specified in
oscars.properties.
  Leave blank for "oscars":
    Using oscars .
- Input the password for the Policy Service account:
    Policy Service account access verified.
```

Enter the username and password of the MySQL account that the Policy Service will use to connect to the database. If you are installing on the same machine as the OSCARS IDC and/or NotificationBroker then you MUST use the password specified in oscars.properties. After entering these values you are ready to build the database:

```
- Got all information. Press return to create the
databases...
    Creating databases oscars_policy
    Initializing databases...
    Databases initialized...
    Granting privileges to Policy Service account...
    Policy Service account authorized.
```

Output such as that shown above indicates success. If you get an error then you should verify that the username and password provided are correct. After building the database the next step is to compile the Policy Service software:

```
- Press return to build Policy Service...


--- Cleaning before compile...
Buildfile: build.xml

clean:

BUILD SUCCESSFUL
Total time: 1 second


--- Compiling...
Buildfile: build.xml

...

BUILD SUCCESSFUL
Total time: 10 seconds


--- Policy Service built.
```

```
##########################################################
#####################

Policy Service built. Please run do_install.sh to
complete this installation.

##########################################################
#####################
```

Some of the output has been omitted from this document but a message such as that above indicates success. The next step is to run the do_install.sh script that installs the policy service on your file system. The policy service will be installed in one of two locations depending on your system configuration:

1. If the OSCARS_HOME environment variable is NOT set then it will be installed in the current directory (dcn-software-suite-0.5.X/policy).
2. If the OSCARS_HOME environment variable is set then the policy service will be installed in OSCARS_HOME. You should run the policy service from OSCARS_HOME if this is the case. This has the added advantage that a machine running other OSCARS services can start and stop the policy service with the `oscars.sh` script.

The output of this step is shown below:

```
% ./do_install.sh
Buildfile: build.xml

...


BUILD SUCCESSFUL
Total time: 3 seconds
Buildfile: build.xml

...

BUILD SUCCESSFUL
Total time: 2 seconds

##########################################################
#####################

Policy Service installed.

##########################################################
#####################
```

If you see the output above then congratulations! You have successful installed the policy service.

3.2 Starting the Policy Service

The policy service consists of two important processes: Tomcat used to accept incoming requests and the Policy core that processes those requests. You may start Tomcat with the following command:

```
% $CATALINA_HOME/bin/startup.sh
```

After starting Tomcat you may start the policy core (*NOTE: You may also start the core first as the order of startup is not significant*). If you **have NOT set the OSCARS_HOME environment variable** then run the following commands to start (or restart) the policy core in daemon mode:

```
% cd dcn-software-suite-0.5.X/policy
% ./policy-core.sh –d
Policy core started.
```

If you **have set the OSCARS_HOME environment variable** then run the following commands to start (or restart) the policy core in daemon mode:

```
% $OSCARS_HOME/policy-core.sh -d
Policy core started.
```

NOTE: If OSCARS_HOME is set then all future calls to `$OSCARS_HOME/oscars.sh` will restart the policy service in addition to the other OSCARS services.

At this point you are ready to proceed with defining your first policy as described in section **4 Defining Policy**.

3.3 Stopping the Policy Service

You may stop and start Tomcat and the core independently. If you need to stop Tomcat then run the following command:

```
% $CATALINA_HOME/bin/shutdown.sh
```

If you **have NOT set the OSCARS_HOME environment variable** then you may run the following command to stop the core:

```
% ./policy-core.sh stop
```

If you **have set the OSCARS_HOME environment variable** then you may run the following command to stop the core:

```
% $OSCARS_HOME/policy-core.sh stop
```

3.4 Configuring the OSCARS IDC to use the Policy Service

If you are running version 0.5.X of the OSCARS IDC then you can configure it to use the Policy Service for incoming createReservation and modifyReservation requests. This allows the IDC to enforce policy based on the decision that the Policy Service returns. Configuration requires you to login into the IDC machine, open `oscars.properties` in a text editor, and add the following lines:

```
policy.useService=1
policy.service.url=https://policy-service-
address:8443/axis2/services/OSCARSPolicy
default.domain=mydomain.net
```

The *policy.service.url* property should be set to the URL of your policy service. If it is running on the same physical host as the IDC and running HTTPS of port 8443 (the default for Tomcat) then it would look like the following: https://127.0.0.1:8443/axis2/services/OSCARSPolicy. Also, the *default.domain* property should be set to the local domain’s topology identifier. It tells the policy service the home domain of local users. After setting these properties you need to restart your IDC with the `oscars.sh` script.

4 Defining Policy

The OSCARS Policy Service currently allows policy to be defined in a configuration file. This section describes the format of a configuration file and how to load its contents into the policy service.

4.1 Policy Configuration File

Policy configuration files define classes of users and the rules that apply to members of those classes. Below is a sample configuration file taken from `dcn-software-suite-0.5.X/policy/conf/example-policies/simple.conf`:

```
#DENY all requests that don't belong to a policy class
policy-class default{
    rule-set default deny;
}

policy-class simpleClass{
    members{
        name "CN=Alice, OU=OASIS Interop Test Cert,
O=OASIS";
        name "bob@mydomain.net";
        role "OSCARS-user"
    }
    rule-set simpleRules allow;
}

rule-set simpleRules{
    match-all{
        match-any{
            action == createReservation;
            action == modifyReservation;
        }
        bandwidth < 500;
    }
}
```

The configuration above defines two policy classes: the *default* class and one named *simpleClass*. The *default* class is a special class that matches any user that does not belong to another class. In our example the default class denies any request. The class named *simpleClass* matches users alice, bob, or those with the OSCARS-user role attribute. Requests from members of this class are only allowed if they match the rules defined in *simpleRules*. The rules defined state that a request must be for a createReservation or modifyReservation action and that the bandwidth of the affected reservation must be less than 500Mbps. This configuration file will be explored in greater detail in the remainder of this section.

4.2 Policy Classes

A **policy class** defines a group of users and the rules that should be applied to them. A policy class is defined with the syntax `policy-class` *`name`*`{…}` where *name* is a value that identifies the class. In the section 4.1 example we defined a policy class named *simpleClass* as follows:

```
policy-class simpleClass{
    ...
}
```

The name may be any valid string with the exception of a special reserved name: *default*. The *default* class is a special class that matches any policy request about a user that does not belong to another class. Since the *default* class is a "catch-all" we don't have to define the members of the class. For all other classes we do need to define the members. This is specified by including a `members` block inside a `policy-class` definition. In *simpleClass* we defined the following:

```
members{
```

```
            name "CN=Alice, OU=OASIS Interop Test Cert, O=OASIS";
            name "bob@mydomain.net";
            role "OSCARS-user"
        }
```

Each line in the block specifies a name or attribute of a user. If a particular user matches ANY of the criteria then the policy for the class will be applied. Currently only the following attributes are allowed:

1. `name` – Either the X.509 subject or OSCARS login of a user. If the user account is managed by a local OSCARS AA service then the policy service will be capable of looking up either the X.509 subject or the login. This means you only have to specify one of these values and if either is used it will match to the particular class. If a user account lives in a remote domain it is recommended you use the X.509 subject as that is how user identifications are currently passed between domains by default. If the user in the remote domain does not have an X.509 certificate you may use the form *login@home-domain*. Also, you must always qualify local users with the local domain (*i.e. login@mydomain.net*) when using the OSCARS login so it is clear who is being referenced.

2. `role` – This matches the role attribute of a user as defined in the local OSCARS AA service. Common examples include “OSCARS-user”, “OSCARS-engineer”, and “OSCARS-service”.

In our example there are three ways a user may be a member of *simpleClass*: by holding the X.509 certificate with the subject "CN=Alice, OU=OASIS Interop Test Cert, O=OASIS", being the owner of the OSCARS login “bob”, or having the “OSCARS-user” role attribute. It should also be noted that a user might belong to multiple policy-classes. For example, if we defined a policy class named *simpleClass2* that include all users with the “OSCARS-engineer” role and user “bob” has that role then “bob” belongs to both *simpleClass* and *simpleClass2*. In such a case “bob” must meet all the policy constraints of BOTH *simpleClass* and *simpleClass2*. We associate policy constraints with a policy class through rule sets. A policy class references which rule-set to use with the `rule-set` directive inside the policy class. The rule-set directive takes the form `rule-set` *`setName`* `allow|deny`. The *setName* indicates the name of the rule-set to use. The rule-set is defined elsewhere in the configuration file as described in section 4.3. The last part of the directive is either *allow* or *deny* and indicates whether a request that matches all the rules should result in a decision to allow or deny the action. Below is the example from section 4.1 for *simpleClass*:

```
        rule-set simpleRules allow;
```

The example defines that any request that matches the rules in the rule-set with name *simpleRules* should be allowed. Alternatively, if we were to specify deny instead of allow then any request that matched the rules in *simpleRules* would be denied. It should also be noted that there is a special rule-set named *default* which points to an empty set of rules. That means that any request that matches a policy class using the *default* rule-set will always be allowed or denied depending on whether *allow* or *deny* is specified. A common practice is to use the *default* rule-set with the *default* policy-class to allow or deny all requests that don't belong to a class. In our previous example the default class denied all requests with the following:

```
rule-set default deny;
```

Those are the basics of defining policy classes in a configuration file. In addition to the features mentioned in this section, policy classes can also be arranged in hierarchies, but that topic is beyond the scope of this section. See section 6.1 for more information on defining policy hierarchies. Proceed to the next section to learn the basics of defining rule sets that apply to policy classes.

4.3 Rule Sets

A **rule set** defines a list of constraints on a particular resource. A rule set is defined with the syntax `rule-set` *`setName`*`{ … }` where *setName* identifies the set. The *setName* also corresponds to the *setName* referenced by the rule-set directive in the policy class definition (see section 4.2). Using the example from section 4.1,a rule set named *simpleRules* is defined as follows:

```
rule-set simpleRules{
     ...
}
```

A rule set contains one of the following blocks at its root:

1. `match-all` – All the rules in this block must be matched
2. `match-any` – any of the rules in this block must be matched

You may nest match-all and match-any blocks to any depth to build your policies. Eventually, though, you will need to define some rules about resources inside of those blocks. Rules take the following form:

```
field operator value;
```

In the example *field* is the name of the field to evaluate, *operator* indicates the operation to perform when evaluating the field (i.e. equals, not equals, etc), and *value* is the value that the field should be evaluated against. Below is the example from sections 4.1:

```
match-all{
     match-any{
          action == createReservation;
```

```
                action == modifyReservation;
            }
            bandwidth < 500;
        }
```

In the example there is a `match-all` block that has two direct descendents. One is a `match-any` block that indicates the action field must be equal to *createReservation* or *modifyReservation*. The other descendent indicates that the bandwidth field must be less than 500Mbps. The OSCARS Policy Service provides a number of fields and operators by default. Fields can be of the following types: strings, numbers, dates or booleans. Depending on the type you have different operators available. The default set of fields should meet most needs related to dynamic circuit provisioning but section 6.2 details how to define custom fields if the default set is not adequate. Below is a description of the various rule types and the operators available.

4.3.1 Strings

String fields are those that can be evaluated as text. They have the following operators available:

Operator	Description
==	Evaluates to true if the value of a field equals the value specified in the rule
!=	Evaluates to true if the value of a field does NOT equal the value specified in the rule
startsWith	Evaluates to true if the field starts with the substring specified by the rule value.
endsWith	Evaluates to true if the field ends with the substring specified by the rule value.
contains	Evaluates to true if the field contains the substring specified by the rule value.

The following string fields are available by default:

Field	Description
action	Matches the action to be performed on the resource. By default the only valid values are

	createReservation or **modifyReservation**.
description	Matches the user-specified description of a reservation.
destination	Matches the URN of the last link in the inter-domain path.
dscp	MPLS-only field. Matches the dscp field.
egress	Matches the URN of the last link in the local path.
gri	Matches the global reservation ID (GRI) of a reservation
ingress	Matches the URN of the first link in the local path.
inter-hop-all	Looks at all hops in the inter-domain path and evaluates to true if all elements match the expression.
inter-hop-any	Looks at all hops in the inter-domain path and evaluates to true if one or more elements match the expression.
inter-hop-*N*	Replace N with a number. Evaluates to true if the hop at position N in the inter-domain path matches the expression.
l4-destination	MPLS-only field. Matches the destination address used to categorize the LSP.
l4-protocol	MPLS-only field. Matches the protocol field. Valid values are “tcp” or “udp”.
l4-source	MPLS-only field. Matches the source address

	used to categorize the LSP.
local-hop-all	Looks at all hops in the local path and evaluates to true if all elements match the expression.
local-hop-any	Looks at all hops in the local path and evaluates to true if one or more elements match the expression.
local-hop-*N*	Replace N with a number. Evaluates to true if the hop at position N in the local path matches the expression.
path-setup-mode	Matches the method of path setup for a reservation. Valid values are **"timer-automatic"** or **"signal-xml".**
source	Matches the URN of the first link in the inter-domain path.

4.3.2 Numbers

Number fields are those that can be evaluated as a numeric value (i.e. 2, 3.55) or range (i.e. "100-200,300-400"). The operators for numeric values are as follows:

Operator	Description
==	Evaluates to true if the value of a field equals the value specified in the rule
!=	Evaluates to true if the value of a field does NOT equal the value specified in the rule
>	Evaluates to true if the field is greater than the value specified in the rule. If a range then all values in the range must be greater than the specified value.

>=	Evaluates to true if the field is greater than or equals the value specified in the rule. If a range then all values in the range must be greater than or equal to the specified value.
<	Evaluates to true if the field is less than the value specified in the rule. If a range then all values in the range must be less than the specified value.
<=	Evaluates to true if the field is less than or equals the value specified in the rule. If a range then all values in the range must be less than or equal to the specified value.

The default fields are as follows:

Field	Description
bandwidth	Matches the bandwidth of a reservation in Mbps.
burst-limit	MPLS-only. Matches the burst limit of a reservation.
dest-vlan	Matches the last VLAN or VLAN range in the inter-domain path
duration	Matches the length of time a reservation may be ACTIVE in seconds (i.e. end-time minus start-time)
egress-vlan	Matches the last VLAN or VLAN range in the local path
ingress-vlan	Matches the first VLAN or VLAN range in the local path
inter-vlan-all	Looks at all VLANs in the inter-domain path and evaluates to true if all elements match the expression.

inter-vlan-any	Looks at all VLANs in the inter-domain path and evaluates to true if one or more elements match the expression.
inter-vlan-*N*	Replace N with a number. Evaluates to true if the VLAN at position N in the inter-domain path matches the expression.
l4-dest-port	MPLS-only. Matches the destination layer 4 port used to assign packets to an LSP.
l4-source-port	MPLS-only. Matches the source layer 4 port used to assign packets to an LSP.
local-vlan-all	Looks at all VLANs in the local path and evaluates to true if all elements match the expression.
local-vlan-any	Looks at all VLANs in the local path and evaluates to true if one or more elements match the expression.
local-vlan-*N*	Replace N with a number. Evaluates to true if the VLAN at position N in the local path matches the expression.
lsp-class	MPLS-only. Matches the LSP class of a reservation.
source-vlan	Matches the first VLAN or VLAN range in the inter-domain path

4.3.3 Dates

Date fields are those that can be evaluated as UNIX timestamps (in seconds). You can specify relative or absolute dates for your comparisons. Also, absolute dates support wildcards that allow you to specify rules that only occur at a particular time of day, month, year, etc. The notation for each date type is below:

- *Relative dates*- Relative dates are wrapped in quotes and start with a + or – sign that indicates whether the time should be added or subtracted to the current time. Immediately following the sign is a value indicating the amount of time to increase/decrease relative to the current time. A space and then a string indicating the units follow this. Valid units are *seconds, minutes, hours, days, weeks, months* or *years*. Examples are below:
  - *"*+2 hours" (2 hours in the future)
  - "-30 seconds" (30 seconds in the past)
  - "+1 days" (1 day in the future)
- *Absolute dates* – Absolute dates explicitly indicate a date and/or time of day in GMT. They take the format ***"YYYY-MM-DD HH:MM:SS"***. Alternatively you can replace any part of the date with a wildcard ('*') to indicate any value matches. Below are some examples:
  - "2010-01-01 00:00:00" (January 1, 2010 at midnight GMT)
  - "2010-01-01 *:*:*" (January 1, 2010 at any time of day GMT)
  - 2010-01-* 10:*:*" (Any day in January 2010 between 10:00:00 and 10:59:59 GMT)

Below are operators that can look at dates. Each operator accepts both relative and absolute values as right operands:

Operator	Description
==	Evaluates to true if the value of a field containing a timestamp equals the date specified in the rule.
!=	Evaluates to true if the value of a field containing a timestamp does NOT equal the date specified in the rule
>	Evaluates to true if the field containing a timestamp is greater than the date specified in the rule.
>=	Evaluates to true if the field containing a timestamp is greater than or equal to the date specified in the rule.
<	Evaluates to true if the field containing a timestamp is less than the date

	specified in the rule.
<=	Evaluates to true if the field containing a timestamp is less than or equal to the date specified in the rule.

The default fields are as follows:

Field	Description
current-time	A special field that evaluates to the current time when the rule is being evaluated. Equivalent to the time a *createReservation* request is made.
end-time	Matches the end time of the reservation.
start-time	Matches the start time of the reservation.

Using the operators and default fields a few more examples are provided below:

- start-time < "+2 weeks" (true if reservation start time is less than 2 weeks in the future)
- current-time >= 2010-01-01 00:00:00" (true if time at which the rule is evaluated is on or after January 1, 2010 at midnight GMT)
- end-time < "*-*-* 12:00:00" (true if reservation ends any day before noon GMT)
- end-time < "*-*-01 12:00:00" (true if reservation ends on the 1st day of any month before noon GMT)

4.3.4 Booleans

A Boolean field is one that evaluates to true (1) or false (0). Currently there are no Boolean fields defined by default. A Boolean field may be defined by creating a custom field as described in section 6.2. The operators available for Boolean fields are as follows:

Operator	Description
==	Evaluates to true the value of a field equals the value specified in the rule. Valid values are 0 or 1.

!=	Evaluates to true the value of a field does NOT equal the value specified in the rule. Valid values are 0 or 1.

4.4 Loading a Policy Configuration File

After defining your policy you need to load it into the policy database with the `policy-config` command. The format of the command is `policy-config` *`filename`*. Below is an example the commands required to run `policy-config`:

```
% cd dcn-software-suite-0.5.X/policy/
% tools/utils/policy-config conf/example-
policies/simple.conf
Policy configuration loaded.
```

It should be noted that in addition to inserting new policies the script will also remove any old policies no longer in the policy configuration file. After this step is complete you have loaded your policy!

5 Logging

If problems occur with your policy service the best place to look is the logs. The logs exist in default locations that you can also modify. Below is a description of where logs are kept and how to change logging behavior.

5.1 Default Logs

Below is a table of relevant logs in the OSCARS Policy Service:

Location	Description
`$OSCARS_HOME/logs/oscars-policy.log`	Contains information related to activity inside the policy engine.
`$CATALINA_HOME/logs/catalina.out`	Contains information related to Tomcat and web service messaging.
`$OSCARS_HOME/logs/oscars-policy-client.log`	Contains information related to the `policy-config` script.

5.2 Changing Log Behavior

Modifying properties files can change the location of logs and amount of logging. The OSCARS Policy service uses Log4J to handle logging so please see [3] for more details. The locations of relevant Log4J properties files are below:

Location	Description
`$OSCARS_HOME/conf/logging/ policy-core.log4j.properties`	Logging related to the Policy Service core.
`$CATALINA_HOME/logs/Catalina.out`	Tomcat logging related to message processing.
`$OSCARS_HOME/conf/logging/ policy-client.log4j.properties`	Logging related to the `policy-config` script

6 Advanced Topics

This section contains advanced topics for those administrators comfortable with the OSCARS Policy Service wishing to know more.

6.1 Policy Class Hierarchies

Policy classes can be arranged in a hierarchical fashion to reduce redundancy in defining policy. When defining a policy class an optional parent policy class can be defined. When a policy class has a parent that means that all the rules that apply to the parent and its ancestors must match in addition to those that are defined just for the child. A policy class may only have one parent. Parents are defined using the `parent` *`parentName`* directive inside a policy class block (*parentName* is the name of the parent class). Below is an example of a policy class hierarchy defintion:

```
policy-class root{
    members{
        user “administrator1”;
    }
    rule-set rules1 allow;
}
```

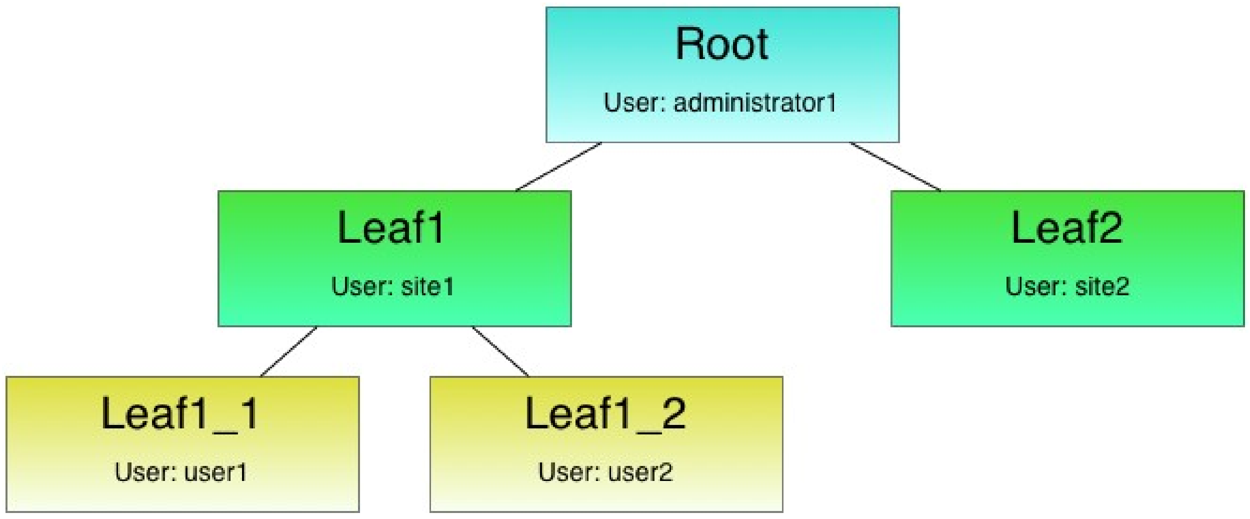
```
policy-class leaf1{
    members{
        user “site1”;
    }
    parent root;
    rule-set “rules2_1” allow;
}

policy-class leaf2{
    members{
        user “site2”;
    }
    parent root;
    rule-set “rules2_2” allow;
}

policy-class leaf1_1{
    members{
        user “user1”;
    }
    parent leaf1;
    rule-set “rules3_1” allow;
}

policy-class leaf1_2{
    members{
        user “user2”;
    }
    parent leaf1;
    rule-set “rules3_2” allow;
}
```

The policy above creates a class hierarchy that looks like the following:



In the example a request by user "site1" would have to adhere to the rules for both the *root* policy class and the *leaf1* policy class. Likewise "user2" would have to adhere to the rules for the *root*, *leaf1*, and *leaf1_2* policy classes. This saves the manager of the policy service the trouble of redefining rules of *root* and *leaf1* while adding a few new ones to *leaf1_2* that don't apply to upper layers. It also means that as rule changes occur at the upper layers they propagate down into the lower level of the hierarchy automatically.

6.2 Defining Custom Rule Fields

You may define custom rule fields by editing the *fields.properties* file. This may be valuable if you'd like to extend the policy service to support resources other than circuit reservations. The location of *fields.properties* depends on your installation. If you have the OSCARS_HOME environment variable set, then it will be under $OSCARS_HOME/conf/properties. If you do not have OSCARS_HOME set, then it will be under $CATALINA_HOME/shared/classes/server. If you open the file in a text editor you will see a number of properties already defined. Below is a snippet from the file that is installed by default:

```
field.string.gri=/idc:reservationResource/idc:globalReser
vationId
field.number.start-
time=/idc:reservationResource/idc:startTime
```

As the example above shows the basic format is `field.`*`type`*`.`*`name`*`=`*`xpath`*. The *type* may be *string*, *number*, *date*, or *boolean*. The *name* of the field indicates how the field will be specified in the policy configuration file. The *xpath* indicates an XPath expression that points to the XML field in a resource to which the name maps. If the XML field contains text it should be of type *string*; if it evaluates to a numeric value it should be of type *number*; if it evaluates to a timestamp(in seconds) it should be a *date*; and if it evaluates to a boolean then it should be *boolean*. See reference [1] for information on the structure of resources and [2] for information on XPath.

There are also some special types of *field* that you can define pertaining to lists of elements. One type indicates that you would like to match a particular element at position N in a list. This can be defined by appending '-#N#' to the end of the field name. For example *inter-vlan-#N#* can be written as *inter-vlan-5* in a policy configuration file to match the fifth VLAN in a list. If no field is found at the position then it evaluates to false. In addition you can say that you want a field to match any single element in a list regardless of position. This can be done by appending '-any' to the end of the field name. For example, *inter-vlan-any* corresponds to any VLAN in the inter-domain path. Rounding out the special list fields you may specify that a field correspond to every element in a list by appending '-all' to the field name. For example *inter-vlan-all* corresponds to all VLANs in the inter-domain path.

You may also define new actions. Since actions map to URNs but URNs are not very user-friendly, field.properties maps shorter names to the action types. The format is `action.`*`name`*`=`*`urn`* where *name* is the short-hand version to be used in the policy configuration file and *urn* is the actual URN.

There is also a special variable you can assign to field: *#current_time#*. It produces a timestamp of the current time when the rule is evaluated and should be used as a date or number type. By default the "current-time" field has this value but you can change the name of the field or create an alias with this variable.

Finally, you may also define new namespaces for XPath with a property such as `namespace.`*`prefix`*`=`*`uri`* where *prefix* is the prefix to use and *uri* is the namespace URI.

7 Further Reading

[1] OSCARS Policy Interface <
https://wiki.internet2.edu/confluence/display/DCNSS/OSCARS+Policy+Interface>

[2] XML Path Language (XPATH) <http://www.w3.org/TR/xpath>

[3] Apache Log4J <http://logging.apache.org/log4j/1.2/>