OpenNMS qosdaemon Documentation

This document describes the qosdaemon which impliments the OpenNMS OSS/J Qos Interface and has been contributed to OpenNMS by the University of Southampton.

Author: Craig Gallen
Version:1.2
Date: 7-11-06

Table of Contents

Introduction to OSS/J

Operational Support Systems through Java (OSS/J) is an initiative initially championed by Sun Microsystems but now encorporated into the Telemanagement Forum Prospero program. (see www.ossj.org and www.tmforum.org). OSS/J provies a means to impliment the TMForum New Generation Operational Support Systems (NGOSS) framework in Java/J2EE environments.

The OSS/J program has developed specifications for a number of Java API's to ease the integation of Telecoms Operational Support Systems. The API's are collaboratively developed using the Java Community process and released as both Java Interface specifications and supporting XSD definitions. Fully implimented OSS/J interfaces support both Java Value Type (JVT) interactions using interfaces defined using an ejb facade pattern and Message Orientated Middleware (MOM) style interfaces using XML messages transported using JMS. Going forwards new API's will be defined with profiles for Web Service (WSDL) style interactions and the Web Services profile will be retrospectively applied to existing interfaces over time.

The earliest and, to date, the most widely deployed of the OSS/J API's are the Quality of Service (Qos) and Trouble Ticket (TT) interfaces. Subsiquent to the initial definition of these interfaces, OSS/J finalised a Common Business Entities (CBE) model in order to provide a more consistant data model which was aligned with the TMForum SID. The newer OSS/J API's closely reference the CBE but the earlier TT and Qos interfaces are not 100% aligned with the current CBE. OSS/J are in the process specifying an aligned Trouble Ticket interface and seperate Fault Management and Performance Management interfaces which will supercede the Qos api. However at the time this project started, the new specifciations had not been released and it was prefered to work with an alreadt finalised specifciation. However the new Fault API is substantially similar to the Qos API and future support of the Fault API should be possible without a major re-write of the interface.. In our design we have chosen not to impliment Qos interface functionality which we know will not be supported by the new Fault interface. (In particular we have avoided using the XmlSerialiser methods which have been deprecieted in OSS/J).

The OSS/J Quality of Service (Qos) specification defines API's for accessing Preformance Management (PM) and Fault Management (FM) data on a Network management System (NMS). It also defines a set of events which the NMS can publish to a Topic using a JMS provider to inform clients of changes in state. The PM and FM sides of the interface can be implimented seperately. The FM interface exposes an alarm list formatted according to the ITU-T X.733 standard and also generates events as JMS messages corresponding changes in the alarm list (for example NewAlarmEvent, ClearedAlarmEvent, AlarmAcknowledgeEvent and AlarmChangedEvent). Clients of the Qos interface can register with the JMS provider to receive only alarms corresponding to certain filter criteria such as PerceivedSeverity or ManagedObjectType or Instance. This allows for a truly distributed system where clients only register to receive messages relavant to their purpose. It is possible for Clients to maintain a local copy of the state of an alarm list using events alone - which avoids the performance hit and potential latency involved in

regularly querying the alarm list. The query interface is required however if the client needs to re-synchronise it's view of the alarm list, say when it is starting up for the first time.

Design Philosophy

The qosdaemon project was initiated with a view to make the OpenNMS project more attractive for Telecoms applications where integration to other OSS systems was an important consideration. As a research project, we also wanted to offer a platform where OSS/J could be demonstrated as providing useful functionality and which would provide a vechacle for the wider community to impliment and experiment with NGOSS type solutions in an Open Source environment.

The JCP process used by OSS/J requires that the specifications be released with reference implimentations and Technology Compatability Kits to test other implimnenations against the specification. Thus the design goals of the OSS/J reference implimentations are to provide complete and accurate implimentations of the OSS/J specifiction but not necessarily to provide re-uable libraries or end user functionality. Also once complete, the OSS/J Reference Implimentations are not necessarily subject to ongoing enhancement or maintainance by the original developers.

By way of contrast, the design goals of this project have been to utilise the OSS/J specifications to realise useful functionality in order to allow practical use cases to be demonstrated. Thus the focus has not been on accuracy or completeness of specification implimentation but on the realisation of a viable end-user use case using OSS/J functionality within a community sustained open source project. It is very important to realise that two key mantras of successful open source projects are; 'Release early and release often.' and 'Make it easy for potential users or contributors to assess and begin using your offering'.

Therefore the design philospphy of the project has been; Firstly, to choose a use case for OpenNMS which would leverage OSS/J and would deliver immediate value to OpenNMS community. Secondly, to contribute the solution in such a way as it is fully sustainable as a mainstream contribution to the OpenNMS project even in it's initial release. Thirdly, to only impliment enough OSS/J functionality as was necessary to support the use case. And finally, to structure the design in such a way as it will be possible for future contributions to go back and complete or address any non-conformances in the design of the interface.

A key aspect of the design has been the seperate packaging of libraries (OSSbeans) which could be generally useful for OSS/J implimentations from the OpenNMS specific interface code (qosdeamon). The OSSbeans libraries are in a seperate Apache 2 liscenced project on Sourceforge. The qosdaemon is fully integrated into the OpenNMS code tree and uses maven to encorporate the OSSbeans dependancies into the opennms build. By this means it will be possible for the development of the OpenNMS and the OSSbeans projects to proceed asynchronously and it will also be possible for other projects to leverage OSSbeans. However the dependence of OpenNMS Qos interface on the OSSbeans libraries should ensure that there is sufficient community interest to sustain and carry forwards the OSSbeans project regardless of other users.

A key advantage of this design approach has been that the discipline of having to integrate into a real application (OpenNMS) has made OSSbeans more useful as a library. Throughout the design we have gone through multiple refactoring steps to partition the functionality so that it can easily be picked up by an external application. A second advantage arises from the fact that designing OSSbeans 2.1.0 as an initial offering has been something of an education. It is possible to see numerous areas where the design could be improved. Thus given sufficient interest, future release of OSSbeans will be able to learn the lessons of the first implimentation in order to provide a more generally useful library. In addition the issues around OSS/J compliance are confined to the OSSbeans library which can have it's own roadmap towards compliance as contributions fill in the gaps.

OSS/J Conformance

This project provides partial implementation of the OSS/J QoS interface for OpenNMS. It is offered as an illustrative and training tool to explain OSS/J and to guage interest from the OpenNMS community in taking the project forwards.

This project leverages the OSSbeans project which provides the core classes for the OSS/J implementation. OSSbeans are a seperate project hosted as part of the University of Southampton OpenOSS initiative at http://sourceforge.net/projects/openoss

The interface is based upon the OSS/J QoS specification available at www.ossj.org. The basic principles and design patterns of the the specification are implemented however not all of the mandatory functionality is complete and the interface has not been tested against the OSS/J SDK.

Where functionality is provided it does so using classes implementing interfaces conforming to the javax.oss interface tree and the XML messaging uses messages conforming to the OSS/J Qos XSD's. This provides a firm basis for moving towards full OSS/J compliance in future releases.

Some work was done previously by the Budapest University of Business and Technology (BUTE) to demonstrate that the PM interface could be implimented for OpenNMS. This work has not been incorporated into the present Qos interface but could be taken forwards later. The present Qos interface only impliments FM functionality. (Note however that it is still possible for OpenNMS performance threshold crossing events to be converted into OSS/J faults reported by openNMS)

Production Use

The interface should be considered experimental and is not optimised for high load environments. Although included with OpenNMS, it can be completely disabled and will not then interfere with

other OpenNMS components. However even in it's present form the interface may still be useful for some production solutions. Envisaged uses include;

* Integration of OpenNMS alarms with other Operational Support Systems using J2EE or JMS
* Monitoring important alarms from remote OpenNMS systems - potentially on a customer's site. (Note that to circumnavigate firewalls JbossMQ can be configured to send JMS messages using HTTP - although this has not been tested.)

As an example, the interface has been successfully used to integrate OpenNMS with an Alarm / Topology correllation engine from Sidonis. www.sidonis.com as part of a proof of concept for managing a Digital TV network

Liscence

The qosdaemon project builds an OSS/J interface for OpenNMS. It is released with OpenNMS under the GPL licence and uses code contributed to OpenNMS by the University of Southampton.

OSSbeans (http://sourceforge.net/projects/openoss) are released under the Apache-2 licence by the University of Southampton.

Functionality Provided

The current release of the qosdaemon module leverages OSSbeans Release 2.1.0 and provides the following functionality. The module provides two daemons which can be used independently or together. The daemons share a data access layer, OssDao which is a data access object which maps the OSS/J Qos interface onto OpenNMS's internal alarm list exposed by the OnmsDao.

The qosd daemon publishes the internal OpenNMS alarm list as an OSS/J alarm list. The qosdrx daemon allows an OpenNMS system to connect using the OSS/J interface to remote OpenNMS systems running qosd. This allows a 'master' OpenNMS to monitor the state of the alarms lists in 'slave' openNMS systems. The present implementation is almost exclusively JMS event driven with limited alarm list query functionality provided as a J2EE option on qosd.

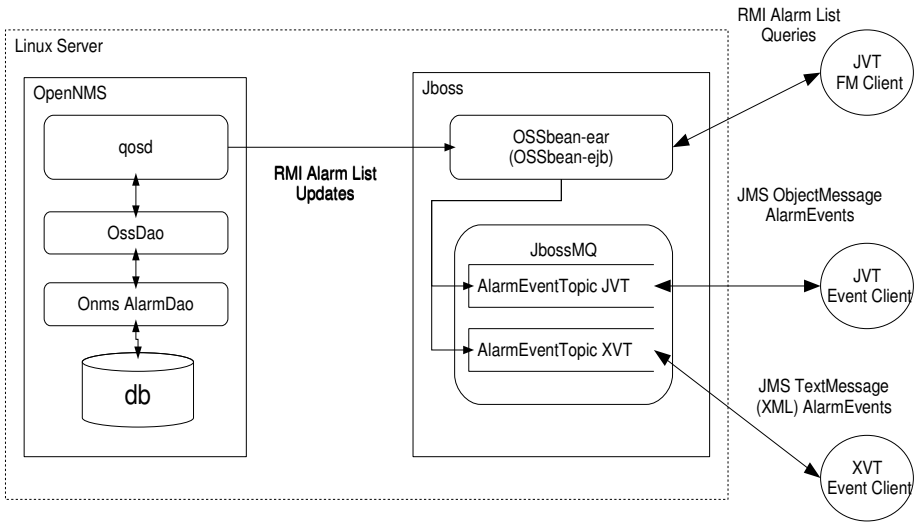
The implementation leverages JbossMQ as the JMS provider. In theory other JMS providers could be used but these have not been tested.

QoSD daemon

The QoSD daemon monitors the OpenNMS alarm list and generates OSS/J JMS events corresponding to changes in the state of the alarms in the list. It can run in two modes; natively on OpenNMS or in conjunction with a separate J2EE application.

a. Native OpenNMS provided interface.

OpenNMS does not run natively in a J2EE container but leverages the spring framework and JMX to provide a container like environment for it's daemons. The qosd daemon code can run natively as a spring application within OpenNMS. In this case it uses the OSS/J XVT (XML over JMS) profile to publish alarm list changes . The qosd daemon publishes OSS/J AlarmEvents as both JMS TextMessages and as JMS ObjectMessages containing AlarmEvent objects.



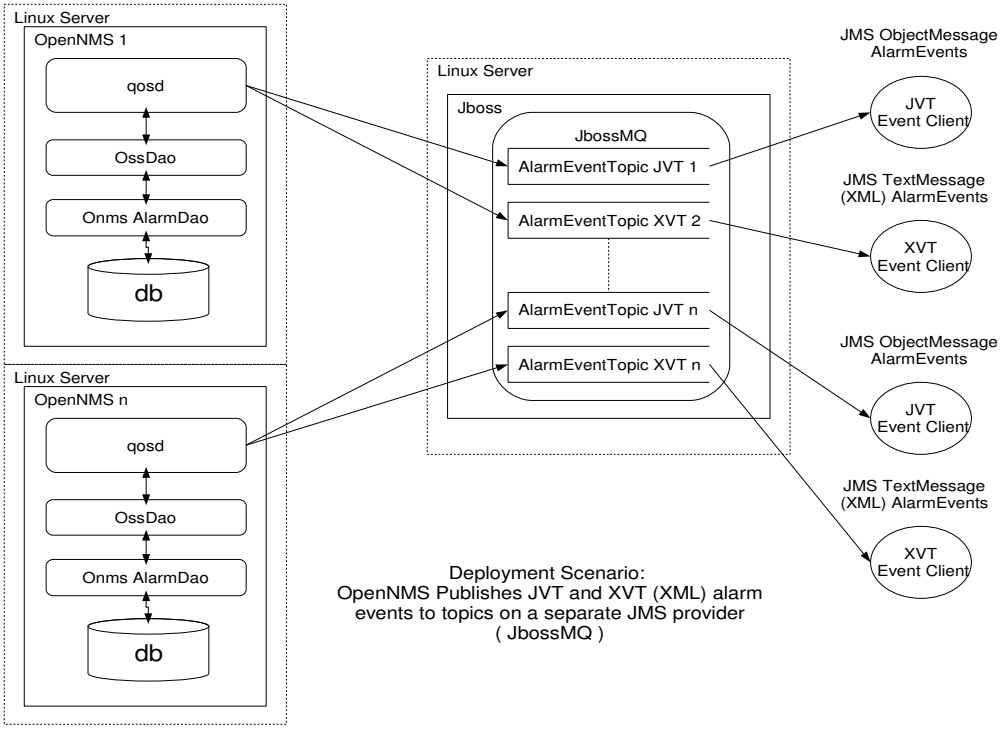
Deployment Scenario :
OpenNMS exposes OSS/J JVT interface using separate J2ee Application

b. Separate J2EE server provided interface.

An alternative configuration is possible where the qosd daemon connects to an ejb application running in a seperate J2EE server. This application is known as OSSBeans-qos-ear and is available from the OSSbeans site. In this mode the ejb exposes OSS/J semantics and allows external applications to connect with the ejb as an OSS/J JVT interface. Note that only a very limited alarm list query functionality is currently provided (query for all alarms).

Note that this configuration requires a J2EE server (Jboss) to be hosting a OSSBeans-qos-ear locally to each OpenNMS implementation which is running qosd in this mode. In most circumstances, it is easier to use the native interface for the remote machines they can all use a single JbossMQ deployment and a local J2EE server is not required for each OpenNMS.

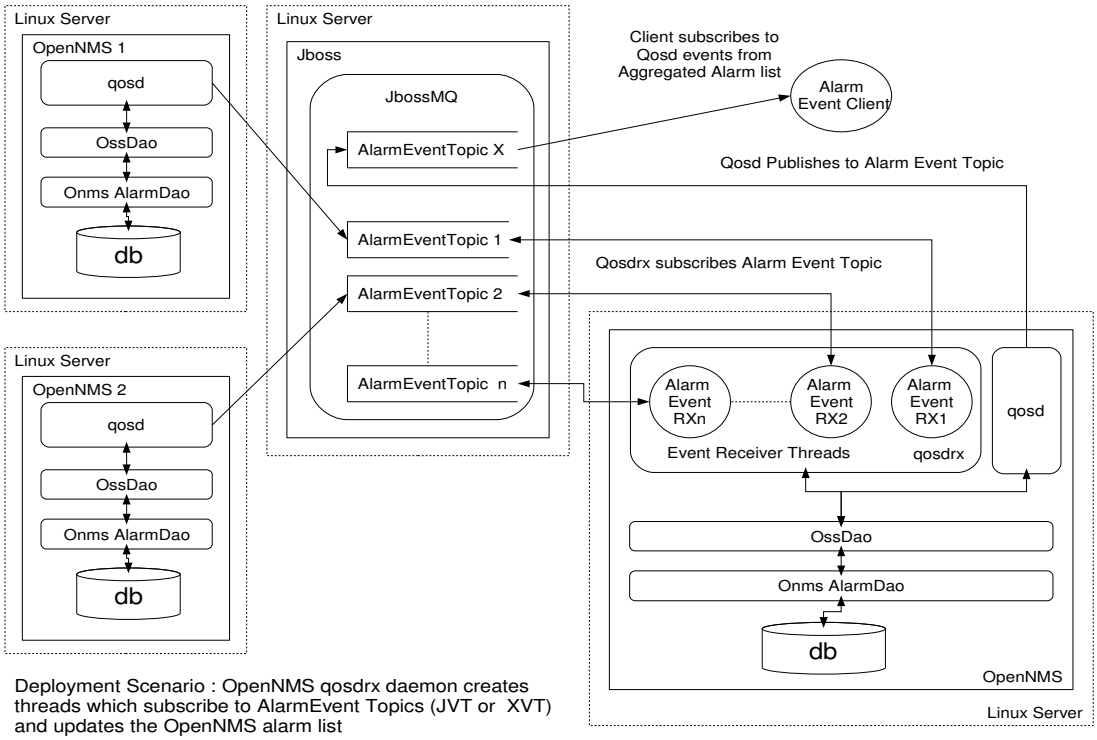
Which mode qosd is running is determined by a setting in the opennms.conf file:
To use the native OpenNMS provided interface use -Dqosd.usej2ee=false
To use the seperate J2EE server provided interface use -Dqosd.usej2ee=true



Deployment Scenario:
OpenNMS Publishes JVT and XVT (XML) alarm events to topics on a separate JMS provider (JbossMQ)

QoSDrx

The qosdrx daemon can connect to multiple OSS/J event topics hosted on a JbossMQ server and receive OSS/J alarm events from remote OpenNMS systems running qosd. The local alarm list will be updated to reflect the remote alarm lists. Note that no resynchronization capability is provided at this time so it is possible for the alarm lists to get out of alignment if messages are lost. However in practice, the JMS messaging system should provide a reliable transport.



Appendix 1: Example Configuration of the OSS/J interface

An example configuration for the OSS/J interface is provided in the OpenNMS /contrib/qosdaemon directory. The following table describes the contents of each of the files.

The simplest way to get the OpenNMS Qos interface working is to build and install opennms of Fedora Core 4 along with tomcat55 and then run the qos instalation script, opennms_1_3_2_example_deploy_xdotx.sh, prior to starting OpenNMS. (See appendix on instalation on FC4 below)

Configuration File	Purpose
/qosdaemon	
README.txt	
LISCENCE(gpl).txt	
/testscrips	
opennms_IF.sh opennms_IFOpenOSS1.sh opennms_IFOpenOSS2.sh opennms_IFOpenOSS3.sh	opennms_IF.sh runs a small client program called SentinalIF.java. This uses the properties in qosclient.properties to connect to the AlarmEvent Topic queue and receive AlarmEvents. The client can display received events and can also forward much simplified XML representation of the X733 Alarm fields to a remove socket for interfacing to other applications. For usage information type sh opennms_IF.sh -help. The opennms_IFOpenOSSx.sh scripts are convenience scripts for starting the same client as opennms_IF.sh using the properties in the qosclientOpenOSSx.properties files)
qosclient.properties qosclientOpenOSS1.properties qosclientOpenOSS2.properties qosclientOpenOSS3.properties	qosclient.properties sets up the configuration for the client interface to connect to anAlarmEvent Topic
/qos_example_configuration	This example configuration provides a simple example of how to set up the OpenNMS qosd application. This should be used to help you work out how to encorporate the qosdaemon into your local configuration.
README.txt	
opennms_1_3_2_example_deploy_1dot0.sh	This script provides a simple method to deploy all of the example configuration files into the correct directories. Notes: 1. This script has been designed for use on a Fedora core 4 installation using a jpackage

	installation of tomcat55. A standard installation of Jboss 4.0.2 is expected to be simlinked from /opt/jboss and OpenNMS is expected to be installed at /opt/OpenNMS. 2. The resulting configuration will leave jboss configured to run on port 8080 and the OpenNMS tomcat at 8081. This is to allow tomcat and jboss to run on the same machine. If they are running on seperate machines, the tomcat setting need not be changed. 2. You must also change the hosts file to create a hostname jbossjmsserver1 pointing to your running jboss server. To do this from the kde toolbar: 3. open /system settings/ network select the hosts tab 4. select new and add a host with Hostname: jbossjmsserver1 Address 127.0.0.1 WARNING: without modification this script will overwrite tomcat55 and OpenNMS configration files in $OPENNMS_HOME/etc so only use on a new install or if you are happy you have backed up your local OpenNMS configruation files
/jboss	This folder contains configuration files to set up JbossMQ messaging (and optionally OSSbeans-qos-ear-xx.ear if it is installed)
log4j.xml	Sets up logging to minimise to INFO log messages from OSSbeans-qos-ear-xx.ear if installed
openoss-jms-service.xml	Sets up 10 separate example AlarmTopics and Message Queues to allow up to 10 OpenNMS qosd deamons to publish to seperate topics. It thould be fairly obvious how to increase / decreae the number of topics or otehrwise change this configuration. Notes 1. The names of the queues match the naming conventions of the OSS/J qos specification. This is merely a convention. The names are actually treated as free format strings. 2. It should be easy to see how the Topic/ Queue names are matched to the names in the qosd.properties, qosdclient.properties and QoSDrxOssBeanRunnerSpringContext.xml
openoss_qos_jboss_start.sh	Starts jboss with the following system settings: -Djava.rmi.server.hostname=jbossjmsserver1 -DqosbeanpropertiesFile= /opt/jboss/server/default/conf/props/qosbean.properties (TODO - this appears not to be used – why?)
uil2-service.xml	Sets up the uil2 transport configuration for JbossMQ.

	This could be changed to set up different JbossMQ transport configuration
/opennms	This folder contains the core configuration files to get the qosdaemon running
opennms.conf	This file is used to set system properties for OpenNMS. The following properties are required for qosdaemon: -Djava.security.policy=/opt/OpenNMS/etc/rmi.policy The security policy is requred to allow OpenNMS to make an rmi connection to the OSSbeans-qos-ear-xx.ear application if installed on jboss. -Djava.naming.provider.url= jnp://jbossjmsserver1:1099 This give the name of the jndi naming provider it is set to jbossjmsserver1 which should be the host name of the Jboss server running JbossMQ. (This name can ofcouurse be changed if all the other configuration references to jbossjmsserver1 are changed) -Djava.naming.factory.initial= org.jnp.interfaces.NamingContextFactory -Djava.naming.factory.url.pkgs=org.jboss.naming Points to the Jboss naming factory (this class is in the Jboss Client library This could in theory be changed to point to another J2ee provider if the correct client classes are available. However this has not been tested) -DpropertiesFile=/opt/OpenNMS/etc/qosd.properties qosd.properties provides the configuration for the JMS topcis which opennms will publish to. -Drx_propertiesFile= /opt/OpenNMS/etc/qosdrx.properties (Note :This configuration is not used as the currnet qosdrx configuration is contained in QoSDrxOssBeanRunnerSpringContext.xml. However qosdrx properties could be used if the commented out example referencing lines in QoSDrxOssBeanRunnerSpringContext.xml are enabled. -Dqosd.usej2ee=false If set false , this property tells qosd to run internally to opennms and publish to the external JbossMQ topics. The interface then only supports AlarmEvents but runs natively in OpenNMS. This is the simplest configuration and recommended for normal use.

	If set true, this property tells qosd to connect to the OSSbeans-qos-ear-xx.ear running in a local Jboss server. This then allows JVT access to the alarm list.
service-configuration.xml	This file is used by OpenNMS to startup it's deamons. To run qosd uncomment the section beginning <service> <name>OpenNMS:Name=QoSD</name> To run qosdrx uncomment the section beginning: <service> <name>OpenNMS:Name=QoSDrx</name> qosd and qosdrx can be run at the same time. However make sure that qosdrx is not configured to listen to the output from qosd – otherwise you will have a very effective positive feedback loop and an ever increaing list of alarms! In this example configuration qosd publishes to Topic .../OpenOSS/... and qosdrx listens to topic .../OpenOSS1/...
QoSD-configuration.xml	This file sets the events which the qosdaemon code will listen to. By default it listens for the event generated by the vacummd configuration when the alarm list changes; uei.opennms.org/vacuumd/alarmListChanged and for events signifying changes in the node inventory which it uses to update it's local managed object instance and managed object type cache: uei.opennms.org/nodes/assetInfoChanged uei.opennms.org/nodes/nodeAdded uei.opennms.org/nodes/nodeLabelChanged uei.opennms.org/nodes/nodeDeleted
QoSDrxOssBeanRunnerSpringContext.xml	This file is a spring application context to set up the configuration for the qosdrx. Seperate threads are set up to connect to each topic. A list of processes to run is in the segment <bean id="OssBeanRunnerList" The definition of each thread is given below. e.g. <bean id="Outstation_OpenOSS1" TODO – full description of this config file
qosdrx.properties	Optional – see note in opennms.conf above

log4j.properties	This file sets up logging for the daemons in openNMS. Two additions are made for QoSD and QoSDrx.
	# QoS daemon server
	log4j.category.OpenOSS.QoSD=DEBUG, QOSD
	...
	# QoSrx daemon server
	log4j.category.OpenOSS.QoSDrx=DEBUG,

	QOSDRX ... Note: In production you should set the logging options in this file to INFO as DEBUG is extremely verbose and will quickly create very large log files and also slow down the QoSD and QoSD deamons significantly..
rmi.policy	Used to allow remote rmi connections to Jboss. (See opennms.conf above). Note that this setting is wide open. You might want to tighten up access security in a production environment.
rrd-configuration.properties	Sets rrd to use jrobin as default instead of rddTool format. (Note that this was needed for the BUTE Qos performance code and not needed for the present qosdaemon)
web.xml	Sets up opennms to use tomcat on port 8081 nd also to refresh the alarm list display at 10 second intervals.
/opennms_fault_config	
eventconf.xml events	Configures how opennms treats incomming traps and events. Note in this configuration this is the same as eventconf_NoOpennmsalarms.xml This file references additional events in /ossj_events.xml
eventconf_WithOpennmsalarms.xml	This is default eventconf.xml with additional ossj events referencing /events/ossj_events.xml
eventconf_NoOpennmsalarms.xml	This is default eventconf.xml with additional ossj events but without the opennms generated alarms. This means the alarm list only contains alarms which have been generted as the result of traps
vacuumd-configuration.xml	VERY Important. Contains automation which looks for new alarms in the alarm list and calls the Qosd daemon using the new uei.opennms.org/vacuumd/alarmListChanged event when new alarms are found. Also reconciles raise with clear alarms and deletes cleared and acknowledged alarms
/events	Contains additional event definitions referenced by eventconf.xml
ossj_events.xml	Contains the ossj specific events created for the qosdeaemon interface and some test snmp trap definitions to allow the scripts in /testtraps to work
/testtraps	
ossjtesttraps_raise.sh	Simple script to raise an alarm on node 127.0.0.1 Alarm raised is definred in ossj_events.xml <uei>uei.opennms.org/ossjTestEvent/newAlarm/1</uei>

ossjtesttraps_clear.sh	Simple script to raise an alarm on node 127.0.0.1 alarm cleared is defined in ossj_events.xml <uei>uei.opennms.org/ossjTestEvent/clearAlarm/1</uei>
trapgen	Simple utility for generating traps from scripts. Written by http://www.ncomtech.com/trapgen.html Note to run this on fc4 you must also have the library compat-libstdc++-296-2.96-132.fc4.i386.rpm installed
/tomcat55	
server.xml	Sets up tomcat to run on port 8081
tomcat55.conf	Various setting to get tomcat55 running on fedora core 4 with opennms; sets JVM, Tomcat user=root etc
/images	The images directory was used for the BUTE performance management application. Not presently used.
OSS_logo_final.gif	
WEB-INF	
web.xml	

Appendix 2: Notes on setting up qosdaemon on Fedora 4

These notes are intended to provide some help in getting OpenNMS runnning with the qosdaemon on Fedora Core 4. They can be adapted to other distributions.

Installing OpenNMS 1.3.2-SNAPSHOT

The following summarises the instructions for installing OpenNMS Fedora 4 based on http://www.opennms.org/index.php/Building_OpenNMS

preliminary set up of FC4

1. It is usefull to have yumex installed as a visual package manager. This makes it easier to pick packages you need for the next steps. (yum install yumex)
2. Ensure subversion is installed on your machine (yum install subversion)
3. Ensure java 1.5 is installed on your machine. This is best done on Fedora core using Jpackage which allows you to easily download and install java packages from the jpackage repo.The following note explains how to use jpackage to install sun java 1.5 on fedora core http://fedoranews.org/mediawiki/index.php/JPackage_Java_for_FC4
4. After installing jpackage, the jpackage repo and java 1.5, install tomcat55 using yumex (do this after jpackage repo is installed so that tomcat gets the right dependancies). Start up tomcat to check it is running and has correct dependancies before proceeding;
   sudo /sbin/service tomcat55 start
   browse to http://localhost:8080 - if tomcat splash is displayed you are in business
   stop tomcat before proceeding; sudo /sbin/service tomcat55 stop
5. Ensure postgres is installed (yum install postgresql)
6. Ensure maven2 is installed on your machine (Note that maven2 is inclueded in the opennms build and opennms will build without it installed. However we need maven2 if you want to build the OSSbeans package seperately.
   a) download maven 2.0.4 from http://maven.apache.org/
   b) unpack into /usr/local/maven-2.0.4/
   c) place the maven /bin directory on your classpath and set up JAVA_HOME to point to your java 1.5 jre. On Fedora core 4 the following .bashrc login script sets this up when you login;

```
# .bashrc
# User specific aliases and functions
PATH=$PATH:/usr/local/maven/maven-2.0.4/bin
export PATH
export JAVA_HOME=/usr/lib/jvm/java
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

d) after setting up the classpath check maven is installed correctly by typing mvn --version

Installing OpenNMS (this is a standard install of opennms on fc4)

7. cd to directory where you want to download and build opennms source code and issue the command;
   svn co https://svn.sourceforge.net/svnroot/opennms/opennms/trunk opennms
8. cd to the downloaded opennms directory (cd opennms)
9. Type the following commands
   sh build.sh clean # this should be done before any build to clean out any previous data
   sh build.sh install -Dopennms.home=/opt/OpenNMS package assembly:attached
   This command will use maven to build opennms. On first use it will take some time as it will download all of the dependancies into a local maven repository on your machine ($home/.m2).
   Note: once the dependancies are downloaded, you can do a build offline using
   sh build.sh -o install -Dopennms.home=/opt/OpenNMS package assembly:attached
10. this builds a zip file called opennms-1.3.2-SNAPSHOT.tar.gz in the target directory. Copy this zip file to /opt/OpenNMS and unpack the contents (need to be root to do this)
11. Follow the instructions in the opennms config guide in /opt/OpenNMS/docs/install.html to set up the postgres database if it is not already set up.
12. Set up the open nms application using following steps;
    a) cd /opt/OpenNMS/bin
    b) make the scripts runnable; sudo chmod 777 *
    c) point openNMS at the jvm; sudo sh runjava -s
    d) upgrade /install the database tables;
       sudo sh install -disU
       (this will upgrade an existing database or install a new one if needed)
    e) link tomcat55 to the opennms web app;
       sudo sh install -y -w /usr/share/tomcat55/conf/Catalina/localhost
13. Start open nms and tomcat and check all is running;
    to start opennms
       cd /opt/OpenNMS /bin directory ; sudo sh opennms -v start
    the consol should show all of daemons starting up correctly
    start tomcat;
       sudo /sbin/service tomcat55 start
    browse to http://localhost:8080/opennms and login using account: admin; password: admin
14. once you are satisfied all is working, shutdown opennms and tomcat before trying to get the qosdaemon to work
    sudo sh opennms -v stop
    sudo /sbin/service tomcat55 stop

running the qosd application

The previous steps provides a standard install of opennms of fc4. Having installed opennms the following steps will allow you to test qosd on your system

15. install jboss 4.0.2

a) jboss provides the jms messaging service. It can be installed on a remote machine but in this example we are installing it locally. Note that the qosdaemon test scripts need some jars from the jboss install to be copied to the /opt/OpenNMS/lib directory. Ideally these whould have been included in the OpenNMS build but unfortunately they are not avaliable in any Maven repository.
b) download jboss 4.0.2 from http://labs.jboss.com/portal/jbossas/download (jboss4.0.2.zip)
c) unpack jboss4.0.2.zip into /opt/
d) create a symbolic link from /opt/jboss/ to /opt/jboss2.0.2 ;
sudo ln -s /opt/jboss-4.0.2 /opt/jboss
e) For this configuration you need to create a host name 'jbossjmsserver1' pointing to this server. To do this from the kde toolbar open /system settings/ network and select the hosts tab. Select new and add a host with Hostname: jbossjmsserver1 Address 127.0.0.1

16. a simple example configuration and install script to get the qosdaemon running can be found in the OpenNMS contrib directory.
a) cd /opt/OpenNMS/contrib/qosdaemon/qos_example_configuration
b) A script is provided to move all of the configuration files into the appropriate directories and get you started.
WARNING: Note - this configuration is provided as an example and will overwrite the opennms default configuration or any other configuration you have installed. It will also change the tomcat55 configuration so that it runs on port 8081 to allow jboss to run on port 8080. (If you will not be running jboss on the local machine the tomcat configuration can be omitted). You can easily adapt this configuration to work with your local configuration but you will need to merge the configuration files appropriately.
c) To install the qosdeamon configuration ;
sudo sh opennms_1_3_2_example_deploy_1dot0.sh
17. Test that the OSS/J test clients work with jboss
a) open a terminal window and move to the Jboss bin directory
cd /opt/jboss/bin
b) run the newly installed startup script;
sudo sh openoss_qos_jboss_start.sh
You should see the jboss consol log. If all is well Jboss will start up .
c) Leave this window open as Jboss will stop if you close it. (You can run jboss as a daemon but this is not covered here). To stop jboss properly type control-c in this window.
d) To see if Jboss is working open a new terminal and try;
telnet jbossjmsserver 1099
You should see something like;

```
Trying 192.168.2.4...
Connected to jbossjmsserver1 (192.168.2.4).
Escape character is '^]'.
.srjava.rmi.MarshalledObject|?
>IhashlocBytest[BobjBytesq~xp?Fur[Txp&?thttp://bitterne:8083/q~q~uq~??sr
org.jnp.server.NamingServer_Stubxr?java.rmi.server.RemoteStub????xrjava.rmi.server.RemoteObject?
a3xpw:
Unicast?
Connection closed by foreign host.
```

e) Use the client test program to connect to jboss
cd /opt/OpenNMS/contrib
sh opennms_IF.sh -xreceive1
You should see something like;

```
***starting sentinal interface program***
***Option: receive - testing OSS/J connection only***
Initialise Session:
Client Properties File Loaded
Using JNP: jnp://jbossjmsserver1:1099
java.naming.provider.url= jnp://jbossjmsserver1:1099
java.naming.factory.initial= org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs= org.jboss.naming
Initial context created
Trying to connect to AlarmMonitorBean
Connecting to AlarmMonitorBean:System/OpenOSS/ApplicationType/AlarmMonitor/Applica
Obtained home, and created Session
Trying to connect to message queues
Topic Connection Factory:System/OpenOSS/ApplicationType/AlarmMonitor/Application/1
Event Topic    :System/OpenOSS/ApplicationType/AlarmMonitor/Application/1-0;0-0;Op
XVT Event Topic:System/OpenOSS/ApplicationType/AlarmMonitor/Application/1-0;0-0;Op
XML Message Queue:System/OpenOSS/ApplicationType/AlarmMonitor/Application/1-0;0-0;
Session Initialised
Subscribing to OSS/J XVT jms session:
Subscribed - Waiting for events ;
Waiting: Event Counts:- ListRebuilt:0   NewAlrmEvt:0    ClrAlrmEvt:0    AlrmCngEvt  ObjMsg:0
TxtMsg:0
***Waiting for OSS/J XVT XML JMS text message event***
```

When OpenNMS starts up you will see the client displaying events and he Event Counts will go up as each event arrives.
To stop the client use Control-c
To see other options for the client try; sh opennms_IF.sh -help
Leave the client running while you start up OpenNMS

18. Test OpenNMS
    a) In the following tests it may be neccessary to first clear the openNMS alarm table of old alarms so you can see what is happening . You can use the postgres gui application pgadmIII to provide an sql terminal to look at the alarm table. Once you have logged

into the OpenNMS database you can use;
delete from alarms; # to delete alarms in table
select * from alarm; # to view all of the alarms in the table.

b) start opennms (/opt/OpenNMS/bin/ sudo sh opennms.-v start). After a short time you should see the following; QoSD and QoSDrx are the two daemons for the qosddeamon interface.

```
OpenNMS.Eventd        : running
OpenNMS.Trapd         : running
OpenNMS.Dhcpd         : running
OpenNMS.Actiond       : running
OpenNMS.Capsd         : running
OpenNMS.Notifd        : running
OpenNMS.Scriptd       : running
OpenNMS.Rtcd          : running
OpenNMS.Pollerd       : running
OpenNMS.Collectd      : running
OpenNMS.Threshd       : running
OpenNMS.Discovery     : running
OpenNMS.Vacuumd       : running
OpenNMS.EventTranslator: running
OpenNMS.PassiveStatusd : running
OpenNMS.QoSD          : running
OpenNMS.QoSDrx        : running
```

Note - the logs for QoSD and QoSDrx are in the /logs library. By default the logging setting is very very verbose. To reduce the log output edit before starting opennms the /opt/OpenNMS/etc/log4j.properties change the lines from DEBUG to INFO;
QoSrx daemon server
log4j.category.OpenOSS.QoSDrx=DEBUG, QOSDRX
QoSrx daemon server
log4j.category.OpenOSS.QoSDrx=DEBUG, QOSDRX

c) Look at the running opennms_IF.sh client. You should see it has received at least an 'AlarmListRebuilt event and possibly others if there were already alarms in the OpenNMS alarms list

d) Open a browser and look at the OpenNMS alarm list at http://localhost:8081/opennms/alarm/

e) Try injecting test alarm raise and alarm clear traps using the following scripts.
cd /opt/OpenNMS/contrib/qosdaemon/qos_example_configuration/testtraps
sudo sh ossjtesttraps_raise.sh
sudo sh ossjtesttraps_clear.sh
You should see alarms added and deleted from the web alarm list as it refreshes
You should also see XML alarm events on the opennms_IF.sh terminal.

19. To test that the qosdrx works you need to install another OpenNMS on a seperate server and change its configuration so that it injects alarms onto the opennms running qosdrx

a) Install and test OpenNMS as above
b) edit /opt/OpenNMS/etc/service-configuration.xml and comment out the section which starts up qosdrx. The remote OpenNMS should not be running qosdra as it will get into a feedback loop.

```
<!--
      <service>
            <name>OpenNMS:Name=QoSDrx</name>
            <class-
name>org.openoss.opennms.spring.qosdrx.jmx.QoSDrx</class-name>
            <invoke at="start" pass="0" method="init"/>
            <invoke at="start" pass="1" method="start"/>
            <invoke at="status" pass="0" method="status"/>
            <invoke at="stop" pass="0" method="stop"/>
      </service>
-->
```

c) Edit /opt/OpenNMS/etc/qosd.properties. Change 'OpenOSS' to OpenOSS1 as below. The OpenNMS qosd will now send alarms to the OpenOSS1 topic;

```
org.openoss.opennms.spring.qosd.naming.provider=jnp://jbossjmsserver1:1099
org.openoss.opennms.spring.qosd.naming.contextfactory=org.jnp.interfaces.NamingContextFactory
org.openoss.opennms.spring.qosd.naming.pkg=org.jboss.naming

org.openoss.opennms.spring.qosd.jvthome=System/OpenOSS1/ApplicationType/AlarmMonitor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/JVTHome

org.openoss.opennms.spring.qosd.jms.topicconnectionfactory=System/OpenOSS1/ApplicationType/AlarmMonitor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/TopicConnectionFactory
org.openoss.opennms.spring.qosd.jms.topic=System/OpenOSS1/ApplicationType/AlarmMonitor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/JVTEventTopic

org.openoss.qosd.jms.xvttopic=System/OpenOSS1/ApplicationType/AlarmMonitor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/XVTEventTopic

org.openoss.qosd.jms.queueconnectionfactory=System/OpenOSS1/ApplicationType/AlarmMonitor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/QueueConnectionFactory
org.openoss.qosd.jms.messagequeue=System/OpenOSS1/ApplicationType/AlarmMonitor/Application/1-0;0-0;OpenNMS_OpenOSS_AM/Comp/MessageQueue

org.openoss.opennms.spring.qosd.url=http://OpenOSS1:8081/opennms
```

d) To test if this OpenNMS is sending messages correctly open a new terminal and start a client to listen to its queue. For convenience this can be done using the script
   cd /opt/OpenNMS/contrib
   sh opennms_IFOpenOSS1.sh -xreceive1
e) Start the second opennms you should see it start up with only the qosd daemon running . You should also see the opennms_IFOpenOSS1.sh terminal register an alarm list rebuilt event
f) Try injecting alarms using the trap scripts above into the second opennms. You should see the first opennms alarm list match the changing remote opennms alarm list.

This completes the installation and testing of the Qos interface. See the sections above for more information on how to configure the interface in you environment.