GROUNDWORK
OPEN SOURCE SOLUTIONS

Feeding, Collecting and Normalizing data into the Foundation 1.1 Framework.

A Developer's Guide for Integrating Monitoring Data Into the Foundation Data Store

Date	***Comments***	***Author***
February, 28 2006	Extended tutorial with Overview Initial review	Roger Ruttimann Thomas Stocking

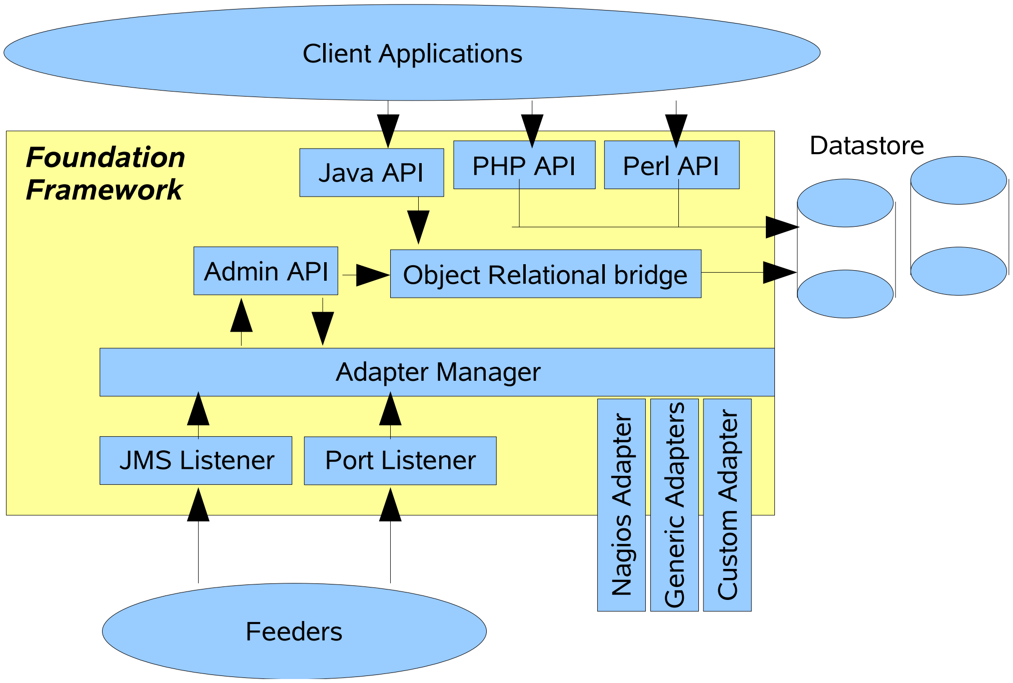
Table of Contents

Foundation Architecture an Overview

The Foundation framework consists of five main components:

1. Feeders – Scripts or programs generating a data set sent to the Foundation listener. The protocol is a simple XML stream.
2. Foundation Listener – A port or JMS Listener receiving the XML streams from various Feeders and dispatching them to data normalizers, called Adapters.
3. Foundation Adapters – Adapters are programs within the Foundation framework that apply rules and data normalization to incoming data. An Adapter calls into the Foundation Admin API for inserting and updating objects. Each Adapter is application specific (Example NagiosEvent, SNMP or Syslog) and is easily added and managed with the framework.
4. Foundation Persistence service – Relational Persistence layer which runs on the top of a database or a database cluster.
5. Foundation API – Documented API's for Java, PHP and Perl, used to retrieve data from the data store

The following diagram shows the interaction between the components:



Dataflow

The data flow of messages is unidirectional, since the Foundation Framework doesn't reply to the incoming XML streams. The data flow is as follows:

FEEDER —> Foundation Listener ---> Adapter ---> Admin API --->Data Store

Component Details:

Feeder

In order to integrate data into the Foundation framework, the data generated by the source application (for example: Nagios or JMX Service) need to be read and sent as an XML stream to one of the listeners. This functionality is provided by *Feeders*. The Feeder can be written in any language. For example, Nagios feeders are written in perl. The xml output protocol is simple:

<FeederName AttributeName='AttributeValue' AttributeName='AttributeValue' … />

The FeederName matches with the Adapter name, and the Attributes are just a list of name value pairs. For example, the Nagios Event Feeder XML has the following format:

<NAGIOSLOG MonitorServer='localhost' Severity='HIGH' TextMessage='Failed to check Host' />

The feeder could include the logic for normalizing the data, but this is discouraged. The best approach is to have a simple and generic feeder that just reads and forwards the data to the listener. Normalization functions are best performed by Adapters.

Foundation Listener

The listener is a simple service either listening on port 4913 or on a JMS topic. The incoming XML message is analyzed and forwarded to the appropriate adapter as defined in the XML element, i.e. the Adapter that matches “FeederName”.

Adapters

Adapters are data normalizers that apply normalization or simplification rules to the incoming XML message. For example, an adapter could calculate the average temperature for a data feed of 10 sensors in a server room, and just insert the calculated value into the data store.

An Adapter can be used for validating incoming data for completeness. It can and should be used to reject incomplete or faulty data before it gets rejected by the persistence layer, which affects system performance.

Adapters are written in Java and compiled into a jar library package. The package includes a Spring assembly file which is read by the Foundation Framework at initialization time. See the tutorial at the end of this document for more details about the syntax of the assembly file, and how to deploy an Adapter.

Data integration approaches

Before you start integrating data into the Foundation data store you have to decide:

- How to collect data from the source application and how to write the feeder.
- Where the data normalization takes place (Feeder or Adapter)
- Whether the default fields in the data model are enough to store your data, or whether you need to add application specific properties.
- What ApplicationType to use for your data. The ApplicationType is a parameter that allows you to access your data using a simple filter. In GroundWork Professional, this filter is built in to the Console View, and will show up automatically when data is present with the application type in question.

Feeder and Generic Adapters

As mentioned in the earlier chapters the Feeder has to produce an XML stream that can be sent to one of Foundation's Listener services. If the feeder performs normalization, or if the input data is simple and matches the default data model properties, then the Feeder can send the data to the *generic* Adapter. The generic Adapter maps the sent attributes to database properties without validating the values.
A sample of how to feed data to the generic adapter can be found in APPENDIX 3 Creating a Feeder for LOG4J and using the generic log Adapter

Custom Properties

If you decide that your application needs more properties to be stored along with the default Status and Event data fields, the following steps are necessary:

- Add new properties and their type to the PropertyType table
- Associate the properties with the EntityType such as LOG_MESSGAE, SERVICE_STATUS or HOST_STATUS.
- Define an ApplicationType for your data.

In the current version of GroundWork Foundation the above database operation needs to be executed with SQL statements. These methods will be supported by the next update of the Admin feeder, which will allow dynamic addition of properties via the input stream.

Custom properties can be inserted using the Generic Adapter, but since no consistency checking is applied, message feeds with missing properties will be rejected.

Recommendations

- For large numbers of constant data feeds, implementation of an adapter to validate the incoming data (all required fields available, correct type) is recommended. This will ensure that any error that would cause a transaction rollback (an expensive middle layer operation) can be detected upfront and rejected.
- Feeders should be simple and as generic as possible– just collect data points and send them to the adapter for normalization. This reduces the load of concurrently running feeder processes, which can be inefficient, especially when written in interpreted languages such as perl.

Tutorial: From Theory to Implementation: A Real-World Adapter for SNMP Traps

Foundation 1.1 Adapter development

The following document describes step by step how to develop a new feeder and adapter for inserting custom data streams into the Foundation framework.

The input source will be SNMP traps.

Development environment

The adapters are distributed as Java libraries (jar) and therefore you need the JAVA SDK available for compiling and packaging the adapters.
The tutorial uses ANT and MAVEN as build tools. You can get ANT and Maven binaries from the Apache site.
The tutorial uses the Foundation 1.1 source distribution which is available form http://sourceforge.net/projects/gwfoundation

Before you start

Install and configure the build tools and Foundation 1.1. Make sure that foundations build runs without errors.

Defining the data that needs to be integrated
As mentioned before the following example integrates (collects/normalizes) SNMP trap events.

What's the Application scope?

SNMP trap are treated as separate application and therefore we have to create a new ApplicationType: SNMPTRAP

What's the Entity Type?

SNMP trap messages will be stored in the LogMessage table and therefore the EntityType is LOG_MESSGAE

What fields need to be stored?

Define the attributes that are generated by the application and specify the if required and the default values if not required.

Attribute	***Type***	***Property***	***required / default value***
Host	String	no	required
Severity	Class Severity	no	required
IpAddress	String	yes	required
MonitorStatus	Class Severity	no	same as Severity
ReportDate	Date	no	default set at time inserted
LastInsertDate	Date	no	default set at time inserted. Can be set by SNMP_LOG message
Event_OID_numeric	String	yes	not required, default to null
Event_OID_symbolic	String	yes	not required, default to null
Event_Name	String	yes	not required, default to null
Category	String	yes	not required, default to null
Variable_Bindings	String	yes	not required, default to null
TextMessage	String	no	not required, default to null

Select fields that are application specific?
A list of fields need to be defined that are properties attached to the table defined as the Entity Type. The properties are application specific and are not part of the base table.
For SNMP traps the following fields and types will be stored:

PropertyName	***Type***
ipaddress	String
Event_OID_numeric	String
Event_OID_symbolic	String
Event_Name	String
Category	String
Variable_Bindings	String

Select fields for data consolidation?
This feature reduces to number of equal messages in the LogMessage table. For each insert the consolidation criteria will be applied to the incoming message as long as the flag in the XML stream is set to to a consolidation criteria name (consolidate=”SNMPTRAP”). By default no consolidation criteria is applied. If the consolidation criteria matches the message counter for an existing message will increased and the date fields will be updated as following:

Field	***change***
FirstInserDate	unchanged
LastInsertDate	ReportDate
ReportDate	System (current time)

Fields that have to match before a message gets consolidated:

- OperationStatus, Host, Severity, ipaddress, MonitorStatus, Event_OID_numeric, Event_Name, Category, Variable_Bindings

Once the data set is defined and properties and consolidation criteria are defined the next step will be to update the database Meta data and writing an adapter for SNMP data normalization.

Database updates

The following database updates can be integrated into one database script that ships with the new adapter. It' always a good idea to check for the existence of an entry before attempting to insert in. For readability of the tutorial these steps are documented.

Since a new application was created the Application needs to be added to the database:
INSERT INTO ApplicationType(Name, Description) VALUES ("SNMPTRAP", "SNMP Trap application");

The new properties need to be created and assigned to the ApplicationType and EntityType

```
INSERT INTO PropertyType(Name, Description, isString)  VALUES ("ipaddress", "ipdddress of snmp device", 1);
INSERT INTO PropertyType(Name, Description, isString)  VALUES ("Event_OID_numeric", "Event_OID_numeric", 1);
and so on for all properties...

INSERT INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT
ApplicationTypeID FROM ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE
Name='LOG_MESSGE'),(SELECT PropertyTypeID FROM PropertyType WHERE Name = 'ipaddress'), 1);
INSERT INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT
ApplicationTypeID FROM ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE
Name='LOG_MESSGE'),(SELECT PropertyTypeID FROM PropertyType WHERE Name = 'Event_OID_numeric'), 1);
and so on for all properties...
```

The consolidation criteria needs to be named and the criteria of matching fields is a semi colon separated list

```
INSERT INTO ConsolidationCriteria(Name, Criteria)  VALUES ("SNMPTRAP", "OperationStatus;Host;Severity;ipaddress,
MonitorStatus; Event_OID_numeric;Event_Name;Category;Variable_Bindings”);
```

Writing the Feeder

The feeder captures SNMP trap and send xml formated messages where all the fields to monitor are XML attributes to the Foundation listener component. The Listener listens on a configurable socket for incoming messages. The default is set to port 4913.

For SNMP traps the XML messages looks as following:

```
<SNMPTRAP
    MonitorServerName="localhost"
    Host="cisco2900.itgroundwork.com"
    Severity="Normal"
    MonitorStatus="Normal"
    ReportDate=""”2005-10-25 04:20.44”
   LastInsertDate=”2005-10-25 04:20.44”
    ipaddress="192.168.2.203"
   Event_OID_numeric=".1.3.6.1.4.1.9.0.1"
    Event_OID_symbolic="enterprises.9.0.1"
   Event_Name="tcpConnectionClose"
   Category="Status Events"
   Variable_Bindings="enterprises.9.2.9.3.1.1.1.1:5 tcpConnState.192.168.2.203.23.192.168.2.249.38591:synReceived
enterprises.9.2.6.1.1.5.192.168.2.203.23.192.168.2.249.38591:600
enterprises.9.2.6.1.1.1.192.168.2.203.23.192.168.2.249.38591:70
enterprises.9.2.6.1.1.2.192.168.2.203.23.192.168.2.249.38591:101 enterprises.9.2.9.2.1.18.1:"
   TextMessage="A tty trap signifies that a TCP connection, previously established with the sending protocol entity for the purposes
of a tty session, has been terminated.   5 synReceived 600 70 101 " />
```

Writing an adapter

Creating the Java project

- Inside the expanded Foundation package go into collagefeeder/adapter and create a new folder called snmp.
- Step inside the snmp directory and create a source/java directory
- Create a a new maven.xml file that looks as following:

```
<project default="jar:install"
        xmlns:j="jelly:core"
        xmlns:maven="jelly:maven"
        xmlns:ant="jelly:ant">

        <goal name='distro'>
    <attainGoal name='clean'/>
    <attainGoal name='jar'/>
    <delete dir='./dist' />
    <mkdir dir='./dist' />
    <mkdir dir='./dist/lib' />
    <copy todir="./dist/lib" file="${maven.build.dir}/${maven.final.name}.jar"/>
    <j:forEach var="lib" items="${pom.artifacts}">
        <j:set var="dep" value="${lib.dependency}"/>
        <j:if test="${dep.getProperty('war.bundle')=='true'}">
            <copy todir="./dist/lib" file="${lib.path}"/>
        </j:if>
    </j:forEach>
  </goal>

  <goal name='allBuild'>
  <attainGoal name='distro'/>
  </goal>
</project>
```

- Create a project.xml that looks as following. Note that the version and the libraries are inherited from the main project file:

```
<project>
  <pomVersion>3</pomVersion>
  <extend>../project.xml</extend>
  <id>collage-adapter-snmp</id>
  <name>Groundwork Collage Adapters for SNMP</name>

  <package>com.groundwork.feeder</package>

   <build>
    <sourceDirectory>src/java</sourceDirectory>
     <resources>
      <resource>
        <directory>${basedir}/src/java</directory>
        <excludes>
          <exclude>**/*.java</exclude>
        </excludes>
      </resource>
    </resources>
  </build>


</project>
```

- Create a new package by creating the following directories under src/java: com.groundwork.feeder.adapter.impl

Now the setup of the new java project that will include the SNMP adapter is done. The next step will show what classes need to be implemented.

Classes and method to overwrite

The adapter has to implement the FeederBase interface which is part of the adapter-api.

Class creation:

- Create a new class SNMPTrap inside com/groundwork/feeder/adapter/impl that implements the FeederBase (Note: will be renamed to AdapterBase)
- Implement GetName that returns the adapter name. The name has to match the node name of the XML fragment send to the listener. For snmp traps its SNMPTRAP
- Implement initialize() and uninitialize() for any actions that need to be executed when the adapter gets loaded or unloaded by the framework.
- Implement the method process() that gets called by the framework for each incoming xml stream the is of the adapter name (SNMPTRAP). Into this method goes the normalization code that transforms the XML message to a database call. The main steps are:
    - parse the XML stream and extarct the attributes. Use the utils.getAttributes() method
    - Get the API object by calling into the bean factory
    - Create a properties map and call into the API

Spring assemblies for SNMPTrap adapter
The new SNMPTrap adapter will be loaded into the Spring container. What you have to include into your JAR file is the spring assembly file which has to be created in the src/java/META-INF directory.
Steps:

- Create a new folder META-INF inside your project's snmp/src/java directory
- In META-INF create a file called assembly-adapter-snmptrap.xml. A sample of the whole file can be find in Appendix 2 Spring assembly for SNMPTrap adapter.

Building the adapter package
Inside the adapter/snmp directory execute:

```
maven jar
```

which will create the jar file in the target directory. Executing:

```
maven jar:install
```

will copy the jar file into the local repository.

Installing and configuring the SNMPTRAP adapter

Once the adapter is compiled successfully it needs to be deployed into the listener installation and the adapter.properties needs to be updated with the new adapter entry

- Copy the jar file (collage-adapter-snmp-1.1.jar) into the listener library path (/usr/local/groundwork/collage/feeder/lib)
- Edit adapter.properties for the new adapter as following:
  increment the counter for assemblies
  - `# Spring assemblies`
  - `nb.assemblies = 3`

  Add the assembly name and the Property Bean name
  - `# SNMPTrap Beans`
  - `adapter.assembly3 = META-INF/assembly-adapter-snmptrap.xml`
  - `adapter.properties.assembly3 = SNMPTrapAdapterProperties`

- Start the listener
- Feed data to the listener and verify if the data shows up in the database. Check the log files for any errors.

APPENDIX 1 SQL script for SNMPTRAP Metadata creation

```
# Database changes for SNMPTRAP messages

# Add new ApplicationType for SNMPTRAP

DELETE FROM ApplicationEntityProperty WHERE ApplicationTypeID = (SELECT ApplicationTypeID FROM ApplicationType WHERE
Name='SNMPTRAP') && EntityTypeID = (SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE');


REPLACE INTO ApplicationType (Name, Description) VALUES("SNMPTRAP","SNMP Trap application");

# Add the properties specific to SNMPTRAP

REPLACE INTO PropertyType(Name, Description, isString)  VALUES ("ipaddress", "ipdddress of snmp device", 1);
REPLACE INTO PropertyType(Name, Description, isString)  VALUES ("Event_OID_numeric", "Event_OID_numeric", 1);
REPLACE INTO PropertyType(Name, Description, isString)  VALUES ("Event_OID_symbolic", "Event_OID_symbolic of snmp device", 1);
REPLACE INTO PropertyType(Name, Description, isString)  VALUES ("Event_Name", "Event_Name", 1);
REPLACE INTO PropertyType(Name, Description, isString)  VALUES ("Category", "Category of snmp device", 1);
REPLACE INTO PropertyType(Name, Description, isString)  VALUES ("Variable_Bindings", "Variable_Bindings", 1);


# Assign the SNMP properties to Application Type SNMPTRAP and Entity LOG_MESSAGE

REPLACE INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT ApplicationTypeID FROM
ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE'),(SELECT
PropertyTypeID FROM PropertyType WHERE Name = 'ipaddress'), 1);
REPLACE INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT ApplicationTypeID FROM
ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE'),(SELECT
PropertyTypeID FROM PropertyType WHERE Name = 'Event_OID_numeric'), 1);
REPLACE INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT ApplicationTypeID FROM
ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE'),(SELECT
PropertyTypeID FROM PropertyType WHERE Name = 'Event_OID_symbolic'), 1);
REPLACE INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT ApplicationTypeID FROM
ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE'),(SELECT
PropertyTypeID FROM PropertyType WHERE Name = 'Event_Name'), 1);
REPLACE INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT ApplicationTypeID FROM
ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE'),(SELECT
PropertyTypeID FROM PropertyType WHERE Name = 'Category'), 1);
REPLACE INTO ApplicationEntityProperty(ApplicationTypeID, EntityTypeID, PropertyTypeID, SortOrder) VALUES ((SELECT ApplicationTypeID FROM
ApplicationType WHERE Name='SNMPTRAP'),(SELECT EntityTypeID FROM EntityType WHERE Name='LOG_MESSAGE'),(SELECT
PropertyTypeID FROM PropertyType WHERE Name = 'Variable_Bindings'), 1);

#Create consolidation criteria

REPLACE INTO ConsolidationCriteria(Name, Criteria)  VALUES ('SNMPTRAP',
'Host;Severity;IpAddress;MonitorStatus;Event_OID_numeric;Event_Name;Category;Variable_Bindings');
```

APPENDIX 2 Spring Assembly for SNMPTrap adapter

The following file assembly-adapter-snmptrap.xml needs to be included into the jar package for the SNMPTrap adapter inside META-INF:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

<!--
 List all the BeanID that have implemented the initialize method. The bean
Id's defined as a comma
 separated list will be called during the loading of the assembly
-->

<bean id="SNMPTrapAdapterProperties"
class="com.groundwork.feeder.adapter.impl.AdapterProperties">
      <constructor-arg
type="java.lang.String"><value>adapter.snmptrap</value></constructor-arg>
  </bean>

<bean id="adapter.snmptrap" singleton="false"
       class="com.groundwork.feeder.adapter.impl.SNMPTrap" />

</beans>
```

APPENDIX 3 Creating a Feeder for LOG4J and Using the Generic Log Adapter

Overview

The Foundation Adapters are pluggable modules, written in Java, that normalize data from an Application before it gets inserted into the data store.
Normalization is needed for complex data structures, but a simple generic adapter with several pre-defined fields might be sufficient for many external log messages, including some application logs.

Foundation provides a GenericLog Adapter that can be used to add log data into Foundation without writing any code. The values assigned to pre-defined fields will be stored in the database under a user defined Application Type.

The following example shows how log4j (used by Application Servers) can be fed into Foundation.

Installing and using the Generic Log adapter

The Generic Log adapter is included in all distributions of Foundation.

Definitions

An ApplicationType for the Adapter to log data against must be defined. In this example the ApplicationType will be LOG4J, but this is user defined and may be adjusted to any type.

Generic Adapter input

The following attributes are required as part of the XML feed, otherwise the message will be rejected: ApplicationType, MonitorServerName, Device, Severity, TextMessage

In addition to the required properties the following properties are accepted but optional: Host, MonitorStatus, ReportDate, OperationStatus, ApplicationSeverity, Component, Priority, ServiceDescription, Priority and TypeRule

Notes about some fields:

If OperationStatus is not defined it will be set to Open

If ReportDate is not set the current time (system) will be used

If MonitorStaus is not defined it will be set to UNKNOWN. MonitorStatus defines the color in the console: OK = Green, DOWN = Red, WARNING = Yellow and UNKNOWN = Blue.

If the FeederScript defines values for the Host and ServiceDescription attributes the LogMessage will be linked (via a Foreign Key) to the matching ServiceStatus. This can be used in the GroundWork Monitor Professional UI to link from an Event Message to the ServiceStatus.

If Host is set the LogMessage will be linked to HostStatus

If Host and Service Description are set the message will be linked to an existing ServiceStatus entry.

If the FeederScript defines a value for the Host attribute, the LogMessage will be linked (via a Foreign Key) to the matching HostStatus entry.

System setup

Create a new ApplicationType (LOG4J) in the database:

Stop the listener and update the database with a SQL statement. In GroundWork Monitor Professional, these statements will work:
Stop GwService from the bash command line:

```
# /etc/init.d/gwservices stop
```

MySQL command line:

```
> mysql -uroot -p GWCollageDB
mysql> INSERT INTO ApplicationType (Name, Description) VALUES("LOG4J","LOG4J Events");
mysql> exit
```

Start GWService:

```
# /etc/init.d/gwservices start
```

Feeder

Here is an example of the simplest possible script that can be used to send one message to the Event table:

```
#!/usr/local/groundwork/bin/perl --
#
#       Copyright 2003-2004 Groundwork Open Source Solution.
#       http://www.itgroundwork.com
#
#
#       Unless required by applicable law or agreed to in writing, software
#       distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
#       WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
#       License for the specific language governing permissions and limitations under
#       the License.
#
use IO::Socket;
my $debug =1 ;
my $remote_host = "localhost";
my $remote_port = 4913;
my $socket;
if ( $socket = IO::Socket::INET->new(PeerAddr => $remote_host,
                                     PeerPort => $remote_port,
                                     Proto    => "tcp",
                                  Type     => SOCK_STREAM)
) {
                    my $xml_message = "<GENERICLOG ApplicationType='LOG4J'
MonitorServerName='localhost' Host='dashboard.itgroundwork.com'
Device='dashboard.itgroundwork.com' Severity='WARNING' MonitorStatus='WARNING'
ErrorType='LogRotation' SubComponent='Log4J Integrator' TextMessage='16:15:54,593 [WARN ]
com.groundwork.collage.impl.LogMessageDAOImpl - Consolidation criteria matches with more than one
record. Make sure the criteria is better defined. If the consolidation criteria was turned on after identical
messages were inserted you have to run consolidate existing messages. Contact support for more
information about database maintenance.' />" ;



          print $xml_message."\n\n" if $debug;
          print $socket $xml_message;
          my $xml_message = "<SERVICE-MAINTENANCE command=\"close\" />";
          print $xml_message."\n\n" if $debug;
          print $socket $xml_message;
         }  else {
     print "Couldn't connect to $remote_host:$remote_port : $@\n";
}
```