Table of Contents

Foundation Overview

What It Does

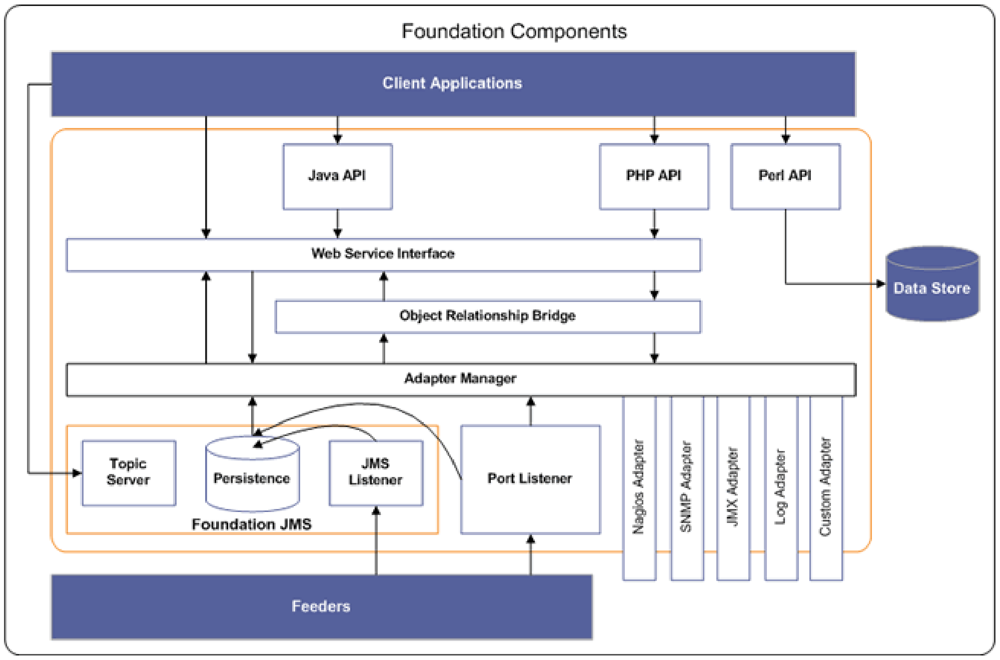
Welcome to the GroundWork Foundation Developer toolkit. Programmers can use the PHP, Perl, or Web Service API or third party products to:

- Build custom displays of real-time monitoring information
- Build custom reports from historical monitoring information
- Monitor additional devices or systems by feeding monitoring information into the system
- Consolidate information from disparate systems into a single view
- Integrate heterogeneous systems by using GroundWork Foundation as an intermediary system

The Developer toolkit includes documentation that describes the available APIs. The toolkit is based on a framework developed by GroundWork Open Source, called GroundWork Foundation 2.0 (*Foundation*). The intent of Foundation is to provide a data model that integrates the components of an IT infrastructure requiring monitoring. Flexible methods of integrating data into the data store are provided allowing different tools and applications and databases to feed data into Foundation. Foundation will normalize the data so it can be retrieved in a consistent manner. Foundation then provides various APIs to allow the normalized data to be retrieved. The Foundation package includes Nagios, as the main monitoring system, integrated with Foundation and a set of applications which use the Foundation APIs to present real-time views and reports.

The Web Service interface is a new addition to the existing Foundation Framework. The previous API components have been re-packaged to use the Web Service Interface and have not been replaced. The following diagram shows the different Foundation components and their interaction:

Figure: Foundation Components



Deployment of Foundation

Foundation is packaged and deployed as a web application (.war) into the Jetty Servlet container. Jetty is an open-source, standards-based, full-featured lightweight servlet container implemented entirely in Java. In addition to the Foundation application the GroundWork implementation includes the following web applications:

- GroundWork Report Server and Eclipse BIRT Viewer that allow to run and manage Eclipse BIRT reprts created with Eclipse BIRT Report Designer.
- GroundWork JMS. A full featured persistence and topic server based on the Open Source project JORAM.
- The professional version of GroundWork Monitor includes the Console implemented as a Web application.

Related Applications

The Monarch tool is a web application that is used to configure the Nagios system. It stores the Nagios configuration data in its own database. At this point in time, the Monarch database is separate from the Foundation database. On a monarch commit the new configuration is synchronized with Foundation.

Storing Nagios Data in Foundation

Since the Nagios monitoring system is integrated in this package, any information gathered by a Nagios plugin can be integrated into the system. Nagios Feeders (in Open Source) or the Event Broker in Professional take the information from the Nagios system and inserts it into the Foundation database. The data objects contained in Foundation map closely to the Nagios objects and include:

Table: Data Objects

Host Groups	This includes Hosts as members.
Hosts	This typically represents a monitoring entity that in Nagios usually maps to physical devices. A Host entry contains one or more Service Checks.
Service	This typically represents a Nagios Service Check for a specific Host. A Host-Service combination is unique in the monitoring instance.

The following type of information can be retrieved:

Table: Data Objects

Host Status	This represents the current status and attributes of Host objects.
Service Status	This represents the current status and attributes of Service objects.
Events	These are typically timestamped messages that are generated by a monitoring system or managed device. The following Nagios Events are stored in the LogMessage table.
Host Alerts	Events generated when a Host changes state.
Host Notifications	Events generated when a notification occurs based on a Host Alert event.
Service Alerts	Events generated when a service changes state.
Service Notifications	Events generated when a notification occurs based on a Service Alert event.
Nagios	Existing Host or Service Alerts are updated when a user acknowledges

Acknowledge	messages.

APIs built on top of the Foundation framework allow this information to be retrieved. The APIs allow programs to query by object and data type. Separate APIs are available for Java, PHP, and Perl programs. In addition to the provided samples, the Foundation status views (Overview, NetView, Troubleview, and FilterView) are built using the PHP Foundation API.

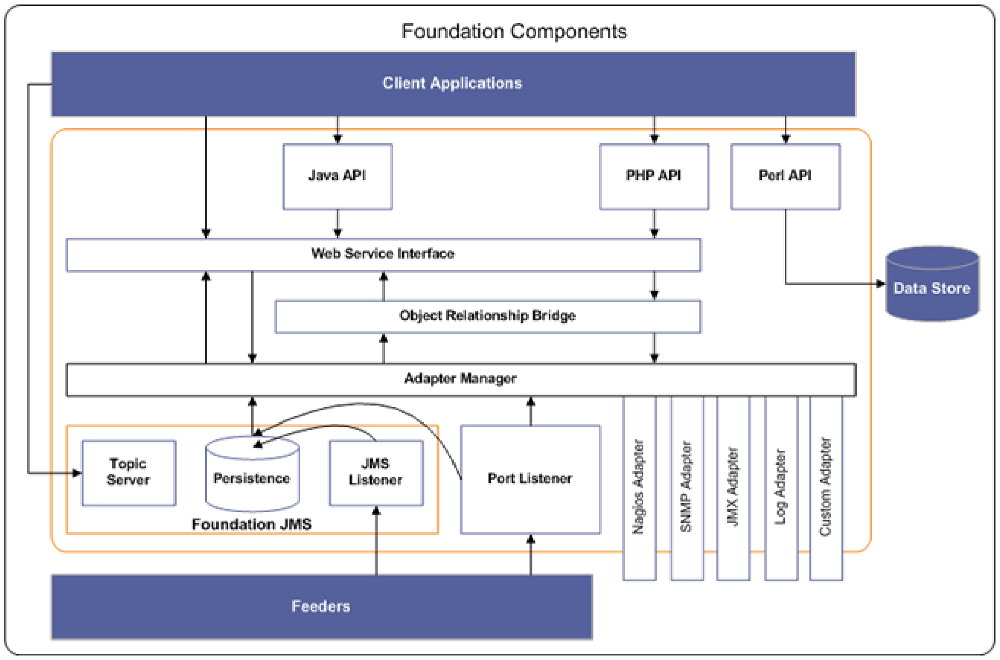
Foundation Architecture and Data Flow

Architecture

This chapter is intended as a developer's guide for integrating monitoring data into the Foundation Data Store. The Foundation framework consists of five main components:

1. **Feeders** These are scripts or programs which generate a data set that is sent to the Foundation Listener. The protocol is a simple XML stream.
2. **Foundation Listener** This is a port or Java Message Service (JMS) Listener which receives the XML streams from various Feeders and dispatches them to data normalizers, called Adapters.
3. **Foundation Adapters** These Adapters are programs within the Foundation framework that apply rules and data normalization to incoming data. Each Adapter is application specific (e.g. NagiosEvent, SNMP or Syslog) and is easily added and managed with the framework.
4. **Foundation Persistence Service** This is a Relational Persistence layer which runs on the top of a database or a database cluster.
5. **Foundation API** This is documented API's for PHP and Perl, used to retrieve data from the data store.

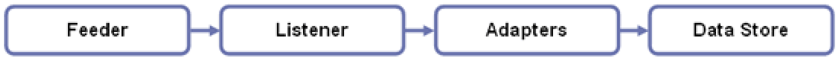
Figure: Component Interaction



Data Flow

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

Figure: Data Flow



Component Details

Feeder

In order to integrate data into the Foundation framework, the data generated by the source application (e.g. Nagios or Java Management Extensions (JMX) Service) needs to be read and sent as an XML stream to one of the listeners. This functionality is provided by *Feeders*. A 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 reads and forwards the data to the Listener. Normalization functions are best performed by Adapters.

The simple format of an XML element represents as well one transaction across the system. For a large load this is expensive and affects the overall message throughput since transaction carry some overhead. The recent version of Foundation includes support for more complex messages where multiple messages can be bundled into one transaction. More details about the different adapters can be found in the "Data Integration approaches section".

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 (e.g., 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 insert the calculated value into the data store.

An Adapter can be used to validate 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 would affect 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 later in this document for more details about the syntax of the assembly file, and how to deploy an Adapter.

Configuring Foundation

Description of the Foundation System

Foundation is a system of several loosely coupled components described below:

- **Foundation-webbapp** - A core component that includes the business objects, the data persistent component (Object Relation Mapping ORM), the data normalizer components (Foundation adapters), and the Web Service API (Soap based API).
- **Foundation-JMS** - The server hosting the Message Queue for incoming data feeds and a Topic server for notification.
- **Foundation-reportserver** - (Professional only) An application to manage BIRT reports.
- **birtviewer** - An Eclipse application to view reports generated with Eclipse BIRT Report Designer.

Each of the components are build as a Web Application and deployed into the servlet conatiner (Jetty). In addition to the web applications Foundation includes the following components:

- Nagios feeders – The nagios feeders read Nagios status and log files and send XML messages to Foundation. The Nagios feeders are only used in GroundWork Monitor Community Edition since the feeder functionality in the Professional version is handled by the Nagios Event Broker. The feeder scripts are located in /usr/local/groundwork/foundation/feeder directory and are named:
    - **nagios2collage_status.pl** - Reads the Nagios status log and updates the Status database with Host and Service status information.
- GroundWork Web Service plugin for Eclipse - The plugin is included in the distribution (usr/local/groundwork/foundation/eclipse) but as well bundled with the birtviewer web component.

Deployment of Foundation

- Foundation Files and Components - /usr/local/groundwork/foundation
- Web Applications - foundation/container/webapps
- Context Files for Web Applications - foundatin/container/contexts
- Configuration for the Jetty servlet Container - foundation/container/etc

Foundation Configuration

Foundation configuration uses properties files stored in /usr/local/groundwork/config. Changing any of these files requires a restart of gwservices as described in the table below.

Configuration File	Description
db.properties	Contains the database credentials for any database used by GroundWork Monitor
foundation.properties	Defines runtime configuration such as the port to listen on, location and configuration of the JMS server and properties to tune the application such as size of the different thread and connection pools.
adapter.properties	List of adapters (Normalizer components) used for message processing.
gwreportserver.properties	Defines the location of the reports and information about the report viewer.
log4j.properties	This file allows to change the level of log reporting for all Java applications. By default is set to Error only. For debugging purposes it can be set to Warning, info or debug.

Logging and Log Output

The output of the log files are defined in log4j.properties. By default all the log files for Java go to the directory /usr/local/groundwork/foundation/container/logs.

Running Foundation

The service to start and stop Foundation is gwservice and is installed into /etc/init.d.

To start foundation issue the following command (needs root privileges):

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

and simiar to stop the service:

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

Note: Caution needs to be applied when starting and stopping services since stopping foundation will shutdown the message listeners and the API driving the User Interface screens.

Data Integration Approaches

Before you Start

Before you start integrating data into the Foundation data store, you need to decide the following:

1. How to collect data from the source application and how to write the Feeder.
2. Where the data normalization takes place (Feeder or Adapter).
3. Whether the default fields in the data model are enough to store your data, or whether you need to add application specific properties.
4. 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 Monitor, this filter is built in to the Console application view, and will show up automatically when data is present with the application type in question.
5. Does it make sense to bundle messages into a single transaction. Bundling would allow a higher message throughput and submit all or nothing of depended data. It adds more complexity to the feeder creating feeds. Foundation added support for this type of messages by defining an XML Schema (link to file) and a new Adapter called SystemAdapter. The recent version of Foundation uses this approach for processing Nagios Status and Event messages.

Supported Adapters

Foundation comes with a set of Adapters for different type of data feeds. The adapters can be classified into two different types:

Single Transaction messages

The adapters of this type accept XML feed of the format <ELEMENT atribute=value,.. />. Each XML element represents a transaction. Foundation supports the following adapters.

Adapter Type	XML	Comments
NAGIOS Events	<NAGIOS_LOG attribute,.. />	Events from Nagios
Nagios Status	<SERVICE_STATUS attributes,./> <HOST_STATUS attribute,../>	Host and Service status updates
SNMP Trap events	<SNMPTRAP attribute,.. />	SNMP trap events coming from the SNMPTT daemon
SNMP Trap events	<SNMPTRAP attribute,.. />	SNMP trap events coming from the SNMPTT daemon
Syslog events	<SYSLOG attribute,.. />	Syslog messages from the gw_syslog plugin
Generic Events	<GENERIC_LOG attribute,../>	Generic adapter that maps attributes directly to dynamic properties without checking.
System messages	<COLLAGE_LOG attribute,../>	Reporting System messages to Console

User definable transaction messages

The preferred way to feed data is using the new SystemAdapter which allows the sending of multiple messages of different entities (Event or Status) in a single transaction. The advantages of this approach are:

- Higher message throughput under load.
- The SystemAdapter uses an XML schema to validate the feed upfront. Any synatx errors are detected before being processed in the service layer.
  - View XML Schema in Firefox: SystemConfig.xsd
  - View XML Schema in Internet Explorer: SystemConfig.xsd
- Wrapping dependent messages in a transaction. If any of the message fail everything will roll back and guarantee data consistency.

For information on how to use and configure feeds that use the SystemAdmin adapter see example in the document Configuring Data Feeders.

The 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 below under *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:

1. Add new properties and their type to the PropertyType table.
2. Associate the properties with the EntityType such as LOG_MESSAGE, SERVICE_STATUS, or HOST_STATUS.
3. 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

1. 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 up front and rejected.
2. Feeders should be simple and as generic as possible - 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.
3. Use the SystemAdapter whenever possible. The performance improvements and the improved message validation make the system much more robust.

Configuring Data Feeders

The following section describes configuring the system for custom data integration and the steps necessary to setup and configure data feeds into GroundWork Monitor for status and event monitoring.

Scenario

An application, let's call it TemperatureWatcher, monitors 10 temperature sensors in different rooms of a building. At startup and shutdown the TemperatureWatcher applications sends events. In normal operations every 30 seconds the temperature of each sensor is sent to GroundWork Monitor.

Additionally, the TemperatureWatcher checks the status of the sensors. If the status is different from an OK status and the sensors do not respond an event is sent to GroundWork Monitor.

The Events generated by TemperatureWatcher should be visible in the Event Console while the temperature of each sensor should be visible in the Status application.

Setup and Configuration

Given the above specifications the following meta data needs to be generated in Foundation:

- **ApplicationType**: TempWatcher
- **Custom Property for Status information**: Temperature of type Double - This is the field where the temperature measurements are stored.
- **Host Group**: TemperatureWatcherApps - The HostGroup needs to be created so that the TemperatureWatcher applications are visible in Status.

Insert Metadata

The easiest way to insert the metadata into Foundation is to stop Foundation, update SQL and restart Foundation.

1. Stop Foundation from the command line logged in as root:

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

2. Update SQL data into the database:

```
mysql -uroot GWCollageDB
```

```
mysql>INSERT INTO ApplicationType(ApplicationTypeID, Name, Description, StateTransitionCriteria) VALUES (200,"TEMPWATCHER", "System monitored by TemperatureWatcher", "Device;Host;ServiceDescription");
mysql>INSERT INTO PropertyType(Name, Description, isDouble)  VALUES ("Temperature", "", 1);
```

3. Restart Foundation:

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

Configuration Data for the Application, Checks and Hostgroup are XML feeds to the TCP port 4913 using the SystemAdmin adapter for Data Normalization. The following Feed creates the necessary entries so that the system can accept monitoring data from the TemperatureWatcher application:

```
<Adapter Session="1" AdapterType="SystemAdmin">
        <Command Action='ADD' ApplicationType='TEMPWATCHER'>
                <Host Host='Temp-Watcher-1' Description='TemperatureWatcher'
                Device='TemperatureWatcher' DisplayName='TemperatureWatcher' />
        </Command>
        <Command Action='ADD' ApplicationType='TEMPWATCHER'>
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_1'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_2'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_3'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_4'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_5'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_6'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_7'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_8'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_9'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_10'
                CheckType='PASSIVE' StateType='SOFT' MonitorStatus='PENDING'
                LastHardState='PENDING' />
```

```
	</Command>
	<Command Action='ADD' ApplicationType='TEMPWATCHER'>
		<HostGroup HostGroup='TemperatureWatcherApps' />
	</Command>
	<Command Action='MODIFY' ApplicationType='TEMPWATCHER'>
		<HostGroup HostGroup='TemperatureWatcherApps' >
		<Host Host='Temp-Watcher-1' />
		</HostGroup>
	</Command>
</Adapter>
```

Feeding Data

Any data (Status and Events) will be sent to the SystemAdmin adapter since the data normalization is done by the feeder application.

Sending an event will use the following XML feed send to the TCP port 4913. The example reports a FATAL error on the sensor_1 check:

```
<Adapter Session='8' AdapterType='SystemAdmin'>
	<Command Action='ADD' ApplicationType='TEMPWATCHER'>
		<LogMessage MonitorServerName='localhost'
		Device='TemperatureWatcher' ServiceDescription='sensor_1'
		TextMessage='Sensor is not reponsing – failed to get temperature'
		ReportDate='2008-01-23 01:45:26' Severity='FATAL'
		MonitorStatus='FAILED' ErrorType='SERVICE ALERT'
		Host='Temp-Watcher-1' />
	</Command>
</Adapter>
```

For best performance the status of all the 10 sensors should be sent as one message to Foundation:

```
<Adapter Session="1" AdapterType="SystemAdmin">
	<Command Action='MODIFY' ApplicationType='TEMPWATCHER'>
		<Service Host='Temp-Watcher-1' ServiceDescription='Sensor_1'
		MonitorStatus='OK' Temperature='76.3' />
		<Service Host='Temp-Watcher-1' ServiceDescription='Sensor_2'
		MonitorStatus='WARNING' Temperature='88' />
		<Service Host='Temp-Watcher-1' ServiceDescription='Sensor_3'
		MonitorStatus='OK' Temperature='43' />
		<Service Host='Temp-Watcher-1' ServiceDescription='Sensor_4'
		MonitorStatus='OK' Temperature='67' />
		<Service Host='Temp-Watcher-1' ServiceDescription='Sensor_5'
```

```
                MonitorStatus='OK' Temperature='76.3' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_6'
                MonitorStatus='OK' Temperature='66' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_7'
                MonitorStatus='OK' Temperature='52.3' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_8'
                MonitorStatus='OK' Temperature='50' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_9'
                MonitorStatus='WARNING' Temperature='93' />
                <Service Host='Temp-Watcher-1' ServiceDescription='Sensor_10'
                MonitorStatus='OK' Temperature='66.3' />
        </Command>
</Adapter>
```

With this configuration Status Data is visible in Status under the HostGroup TempWatcherApp and all Events are visible in the Event Console.

Configuring Consolidation

How It Works

The Foundation consolidation feature allows to reduce the number of LogMessages by creating just one entry for messages that are alike, incrementing the Message counter, and adjusting the date fields. Consolidation processing is applied to all incoming messages of Entity Type LogMessage where the attribute consolidation is defined (e.g. consolidation='SNMPTRAP'.

Consolidation criteria are stored in the Database and consist of a user definable name and the criteria. The criteria is a semicolon (;) separated list of LogMessage record fields or PropertyNames. For an incoming message to be consolidated, all values of the fields defined in the criteria have to match an existing record.

An incoming message must define the Name of the Consolidation criteria that it will be matched against. If the consolidation tag is missing, a new log entry is created, which is the system's default behavior.

Default Consolidation

By default, the Consolidation is turned on for Nagios, SNMP, and Syslog event processing. Consolidation will occur if the fields defined in the consolidation criteria match. An exception to the above is if the Monitoring Status of the last event is different than the Monitoring Status of the incoming event, a new Console message will be created. This rule guarantees that console messages sorted by chronological order will always show the current status above previous status. The following Consolidation criteria are defined in Foundation's ConsolidationCriteria table:

Consolidation Criteria	Used in Feeder	Comments
NAGIOSEVENT	nagios2collage_eventlog.pl	Community Edition
NAGIOSEVENT	EventBroker	Professional Edition
SNMPTRAP	snmptt forwarding traps to Foundation	Professional only
SYSLOG	gw-syslog-feeder.pl	Professional only

Disable Consolidation

To disable consolidation for the Feeders shipped with GroundWork Monitor Community Edition, you can modify the Nagios event log feeder, nagios2collage_eventlog.pl, to send a message to Foundation that does not specify consolidation. Change the script that contains the following line:

my $xml_message = "< NAGIOS_LOG consolidation='NAGIOSEVENT'"; # Start message tag. Consolidation is ON

To the following:

my $xml_message = "< NAGIOS_LOG "; # Start message tag. Consolidation is now turned OFF

Example

To enable consolidation, the event message needs to look like the following, followed by the other arguments:

<NAGIOS_LOG consolidation='NAGIOSEVENT'

The Consolidation Criteria in the database for NAGIOSEVENT defines the following criteria:

Device;MonitorStatus;OperationStatus;SubComponent;ErrorType

If an event is fed to the system and the consolidation criteria is defined, the system checks if any log message for the Device, MonitorStatus, OperationStatus, SubComponent and ErrorType already exists. The system will only consolidate if:

- Just one LogMessage object matches.
- If multiple LogMessage objects match a warning is logged indicating that the criteria needs to be better defined.

If a match is found an existing message the message counter LogMessage.MsgCount for an existing message will incremented and the date fields will be updated as following:

Database Field	**Change Applied**
FirstInsertDate	unchanged
LastInsertDate	ReportDate
ReportDate	System (current time)
TextMessage	Updated Text message. Text Message might include the values of a check (85% disk used) that changes while the status and the type remain the same.

GroundWork Foundation Data Objects and Attributes
Note: The attributes listed below are Nagios specific and apply to the application type Nagios. The Nagios Adpaters (Nagios_log (Events), Host_Status Service_Status) check for the existence of the following attributes.

Host Groups, Device, and Monitoring Server attributes are processed by the SystemAdmin adapter.

Foundation's data store has a flexible data format that can be expanded if needed. The Foundation data objects and attributes implemented in the Foundation package that are applicable to a Nagios-centric are listed below.

Host Status Attributes

- MonitorStatus
- LastCheckTime
- LastStateChange
- isAcknowledged
- TimeUp
- TimeDown
- TimeUnreachable
- LastNotificationTime
- CurrentNotificationNumber
- isNotificationsEnabled
- isEventHandlersEnabled
- isChecksEnabled
- isFlapDetectionEnabled
- isHostIsFlapping
- PercentStateChange
- ScheduledDowntimeDepth
- isFailurePredictionEnabled
- isProcessPerformanceData
- LastPluginOutput

Service Status Attributes

- Host
- MonitorStatus
- RetryNumber
- StateType
- LastCheckTime
- NextCheckTime
- CheckType
- isChecksEnabled
- isAcceptPassiveChecks
- isEventHandlersEnabled
- LastStateChange
- isProblemAcknowledged
- LastHardState
- TimeOK
- TimeUnknown
- TimeWarning
- TimeCritical
- LastNotificationTime
- CurrentNotificationNumber
- isNotificationsEnabled
- Latency
- ExecutionTime
- isFlapDetectionEnabled
- isServiceFlapping
- PercentStateChange
- ScheduledDowntimeDepth
- isProcessPerformanceData
- isObsessOverService

Event - Host or Service Alert

- Host
- ServiceDescription
- Severity
- HostStatus
- ServiceStatus
- TextMessage
- ReportDate
- LastInsertDate
- FirstInsertDate
- SubComponent
- ErrorType

Event - Host or Service Notification

- Host
- ServiceDescription
- Severity
- HostStatus
- ServiceStatus
- TextMessage
- ReportDate
- LastInsertDate
- FirstInsertDate
- SubComponent
- ErrorType
- LoggerName

Host Groups

- Name
- Description

Device

- DisplayName
- Description
- Identification

Monitoring Server

- MonitorServerName
- IP
- Description

Web Service API

Foundation Web Service

Adding a Web Service layer enables more applications to use and integrate with the existing Foundation Framework. The Web Service API uses SOAP (Simple Object Access Protocol) as the communication protocol. Since Web Services are a well defined and a widely used standard other technologies such as .NET or popular development tools such as Visual Studio or Java Studio Creator can be used to create UI or business components on the top of the Foundation platform.

The Web Services framework is able to scale and distribute the API more easily, than the previous API, which results in higher throughput and therefore better overall performance of large installations. The Advanced Reporting feature uses the Foundation Web Service API to access data stored in the Foundation persistent store.

WSDL

Web Service Description Language for GroundWork can be consumed by applications to view and retrieve data from the endpoint (Foundation).

Foundation Web Service Model

Defines the basic types and objects used by the Foundation Web Service.

- fwsmodel.wsdl (Firefox)
- fwsmodel.wsdl (Internet Explorer)

Foundation Event WS

A Web Service used to retrieve Event Messages stored in Foundation.

- fwsevent.wsdl (Firefox)
- fwsevent.wsdl (Internet Explorer)

Foundation Common Web Service

Web Service used to retrieve metadata and access to the Actions Framework.

- fwscommon.wsdl (Firefox)
- fwscommon.wsdl (Internet Explorer)

The following Web Services allow to query for data about a specific Entity Type:

- Device
    - fwsdevice.wsdl (Firefox)
    - fwsdevice.wsdl (Internet Explorer)
- Host
    - fwshost.wsdl (Firefox)
    - fwshost.wsdl (Internet Explorer)
- HostGroup
    - fwshostgroup.wsdl (Firefox)
    - fwshostgroup.wsdl (Internet Explorer)
- Service
    - fwsservice.wsdl (Firefox)
    - fwsservice.wsdl (Internet Explorer)
- Statistics
    - fwsstatistics.wsdl (Firefox)
    - fwsstatistics.wsdl (Internet Explorer)