

Zenoss Administration



Zenoss Administration

Copyright © 2009 Zenoss, Inc., 275 West St. Suite 204, Annapolis, MD 21401, U.S.A. All rights reserved.

This work is licensed under a Creative Commons Attribution Share Alike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>; or send a letter to Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA.



The Zenoss logo is a registered trademark of Zenoss, Inc. Zenoss and Open Enterprise Management are trademarks of Zenoss, Inc. in the U.S. and other countries.

Flash is a registered trademark of Adobe Systems Incorporated.

Java is a registered trademark of Sun Microsystems, Inc.

Linux is a registered trademark of Linus Torvalds.

Oracle and the Oracle logo are registered trademarks of the Oracle Corporation.

SNMP Informant is a trademark of Garth K. Williams (Informant Systems, Inc.).

Sybase is a registered trademark of Sybase, Inc.

Tomcat is a trademark of the Apache Software Foundation.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

All other companies and products mentioned are trademarks and property of their respective owners.

Part Number: 06-102009-2.5-v01

| | |
|---|----|
| 1. About Zenoss | 1 |
| 1.1. Zenoss High-Level View | 1 |
| 1.1.1. Key Tenets | 1 |
| 1.2. Zenoss Architecture and Technologies | 2 |
| 1.2.1. User Layer | 3 |
| 1.2.2. Data Layer | 3 |
| 1.2.3. Process Layer | 3 |
| 1.2.4. Collection Layer | 4 |
| 1.3. Zenoss' Monitoring Approach | 4 |
| 1.3.1. File System Monitoring | 4 |
| 1.4. Zenoss Terminology | 5 |
| 2. Using Zenoss | 7 |
| 2.1. Zenoss Interface and Navigation | 7 |
| 2.1.1. Navigation Menu | 8 |
| 2.1.1.1. Hiding the Menu | 9 |
| 2.1.1.2. Pinning the Menu | 9 |
| 2.1.2. Breadcrumbs | 9 |
| 2.1.3. User Information Area | 10 |
| 2.1.3.1. Update Details | 10 |
| 2.1.4. Portlets | 10 |
| 2.1.4.1. Customizing Portlets | 12 |
| 2.1.4.2. Adding and Duplicating Portlets | 12 |
| 2.1.5. Zenoss Network Map | 12 |
| 2.1.5.1. Choosing the Network to Display | 13 |
| 2.1.5.2. Viewing Device and Network Details | 13 |
| 2.1.5.3. Loading Link Data | 13 |
| 2.1.5.4. Filtering by Device Type | 13 |
| 2.1.5.5. Adjusting Viewable Hops | 14 |
| 2.1.5.6. Adjusting the Network Map | 14 |
| 2.1.5.7. Viewing Device or Network Details | 14 |
| 2.1.6. Zenoss Menus | 14 |
| 2.1.6.1. Page Menus | 14 |
| 2.1.6.2. Table Menus | 14 |
| 2.2. Customizing the Zenoss Dashboard | 15 |
| 2.2.1. Selecting Portlets | 16 |
| 2.2.2. Arranging Portlets | 16 |
| 2.2.3. Changing the Dashboard Column Layout | 16 |
| 2.3. Searching for Devices | 16 |
| 2.4. Navigating the Event Console | 17 |
| 2.4.1. Sorting and Filtering Events | 17 |
| 2.4.1.1. Saving a Custom View | 18 |
| 2.4.2. Refreshing the View | 19 |
| 2.4.3. Viewing Event Details | 19 |
| 2.4.4. Selecting Events | 20 |
| 2.4.5. Managing Events | 20 |
| 2.5. Running Commands from the User Interface | 20 |
| 2.6. Creating and Using Alerts | 21 |
| 2.6.1. Setting SMTP Settings For Alerts | 21 |
| 2.6.2. Creating an Alerting Rule | 22 |
| 2.6.2.1. Define and Enable the Alert | 24 |
| 2.6.2.2. Create the Content of the Alert Message | 24 |
| 2.6.2.3. Create a Schedule for Sending the Alert | 25 |
| 2.6.3. Escalation of Alerting in Zenoss | 27 |
| 2.6.3.1. Creating an Alerting Hierarchy | 27 |
| 2.6.4. Adding Delay and Schedules to Alerting Rules | 27 |
| 2.7. Creating Custom Event Views | 28 |
| 3. Discovering and Modeling Devices | 30 |
| 3.1. How Does Zenoss Model Devices? | 30 |

| | |
|--|----|
| 3.1.1. ZenModeler Daemon | 30 |
| 3.2. Add a Single Device | 30 |
| 3.2.1. Add a Device Within a Device Class | 32 |
| 3.2.2. Add a Device - Alternate Method | 33 |
| 3.3. Discover Devices | 33 |
| 3.3.1. Discover Devices - Alternate Method | 34 |
| 3.3.2. Classifying Discovered Devices | 34 |
| 3.3.3. Authenticating Devices | 35 |
| 3.3.4. Adding Information to a Device Record | 35 |
| 3.4. Modeling Devices Using SNMP | 35 |
| 3.4.1. Testing to See if a Device is Running SNMP | 35 |
| 3.4.2. Modeling Remote Windows Devices Using SNMP | 35 |
| 3.4.3. Modeling Remote Linux Devices Using SNMP | 35 |
| 3.4.4. Modeling Cisco Devices Using SNMP | 36 |
| 3.5. Modeling Using SSH/CMD | 36 |
| 3.5.1. Using Device Class to Monitor Devices Using SSH | 36 |
| 3.6. Modeling Devices Using Port Scan | 36 |
| 3.6.1. Using the /Server/Scan Device Class to Monitor with Port Scan | 37 |
| 3.7. Collector Plugins | 37 |
| 3.7.1. Viewing Collector Plugins for a Device | 37 |
| 3.8. Debugging the Modeling Process | 37 |
| 4. Working with Devices | 38 |
| 4.1. Device List | 38 |
| 4.1.1. Device Page Tabs | 38 |
| 4.1.1.1. Status Tab | 38 |
| 4.1.1.2. OS (Operating Systems) Tab | 40 |
| 4.1.1.3. Hardware Tab | 42 |
| 4.1.1.4. Software Tab | 43 |
| 4.1.1.5. Events Tab | 44 |
| 4.1.1.6. Performance (Perf) Tab | 45 |
| 4.1.1.7. Edit Tab | 46 |
| 4.2. Managing Devices and Device Attributes | 47 |
| 4.2.1. Managing Custom Device Properties | 47 |
| 4.2.2. Managing Device zProperties | 47 |
| 4.2.3. Managing Device Templates | 48 |
| 4.2.4. Determining Device Administration | 48 |
| 4.2.5. Clearing Heartbeats | 49 |
| 4.2.6. Pushing Configuration Changes to Zenoss | 49 |
| 4.2.7. Locking Device Configuration | 49 |
| 4.2.8. Renaming a Device | 50 |
| 4.2.9. Remodeling a Device | 50 |
| 4.2.10. Resetting the Device Manage IP Address | 51 |
| 4.2.11. Resetting the Device Community | 51 |
| 4.2.12. Selecting Device Collector Plugins | 51 |
| 4.2.13. Deleting a Device from Zenoss | 52 |
| 4.2.14. Managing Multiple Devices from the Device List | 52 |
| 4.2.15. Dumping and Loading Devices Using XML | 52 |
| 5. Properties and Templates | 54 |
| 5.1. zProperties | 54 |
| 5.1.1. zProperties Inheritance and Override | 54 |
| 5.1.1.1. User Interface View | 56 |
| 5.1.2. zProperty Types | 57 |
| 5.1.3. Event zProperties | 57 |
| 5.1.4. Device zProperties | 57 |
| 5.1.5. Service zProperties | 60 |
| 5.1.6. Process zProperties | 60 |
| 5.1.7. Network zProperties | 61 |
| 5.1.8. Manufacturer zProperties | 61 |

| | |
|---|----|
| 5.2. Templates | 61 |
| 5.2.1. Template Binding | 61 |
| 5.2.1.1. Device Templates | 61 |
| 5.2.1.2. Component Templates | 62 |
| 5.2.1.3. Interface Templates | 62 |
| 5.2.1.4. Defining Templates in the Device Hierarchy | 62 |
| 5.2.1.5. Applying Templates to Multiple Areas in the Device Hierarchy | 62 |
| 6. Core Monitoring | 63 |
| 6.1. Availability Monitoring | 63 |
| 6.1.1. Controlling the Ping Cycle Time | 63 |
| 6.1.2. Using the Predefined /Ping Device Class | 63 |
| 6.1.3. Monitoring TCP Services | 63 |
| 6.1.3.1. ZenStatus | 64 |
| 6.1.3.2. Adding a Service to Monitor | 64 |
| 6.1.3.3. Monitoring Status Service Status Information | 64 |
| 6.1.3.4. Editing Service Information | 65 |
| 6.1.3.5. Configuring Service zProperties | 65 |
| 6.1.3.6. Using the Predefined /Server/Scan Device Class | 66 |
| 6.1.3.7. Monitoring a Service Using a Service Class | 66 |
| 6.1.4. Monitoring Processes | 68 |
| 6.1.4.1. Adding Processes to Monitor | 69 |
| 6.1.4.2. Configuring Process zProperties | 70 |
| 6.2. Performance Monitoring | 71 |
| 6.2.1. About Performance Monitoring | 71 |
| 6.2.2. Performance Templates | 71 |
| 6.2.2.1. Viewing Performance Templates | 71 |
| 6.2.3. Template Binding | 72 |
| 6.2.4. Data Sources | 73 |
| 6.2.4.1. Adding a Data Source | 73 |
| 6.2.5. Data Points | 74 |
| 6.2.6. Data Point Aliases | 76 |
| 6.2.6.1. Alias Formula Evaluation | 77 |
| 6.2.6.2. Adding a Data Point Alias | 78 |
| 6.2.6.3. Zenoss Reports That Use Aliases | 78 |
| 6.2.7. Thresholds | 79 |
| 6.2.8. Performance Graphs | 80 |
| 6.2.8.1. Graph Points | 81 |
| 6.2.8.2. Custom Graph Definition | 83 |
| 6.2.8.3. Graph Commands | 83 |
| 6.2.9. Changing Graph Display Order | 83 |
| 6.3. Monitoring Using ZenCommand | 84 |
| 6.3.1. About ZenCommands | 84 |
| 6.3.2. Example: Writing a ZenCommand (check_http example) | 84 |
| 6.3.3. Example: Collect Data from A ZenCommand | 85 |
| 6.3.4. Plugin Format for ZenCommands | 86 |
| 6.3.5. Testing ZenCommands | 87 |
| 6.4. SNMP Monitoring | 87 |
| 6.5. Monitoring Devices Remotely Through SSH | 88 |
| 6.5.1. Installing Zenoss Plugins on the Remote Machine | 88 |
| 6.5.1.1. Zenoss Plugin Installation Technique: RPM | 88 |
| 6.5.1.2. Zenoss Plugin Installation Technique: setuptools | 88 |
| 6.5.1.3. Testing the Plugin Installation | 88 |
| 6.5.1.4. Troubleshooting Plugin Installation | 89 |
| 6.5.1.5. Changing Zenoss to Monitor Devices Remotely Using SSH | 89 |
| 6.5.1.6. Using the Predefined /Server/Cmd Device Class | 90 |
| 6.6. Monitoring Windows Devices | 90 |
| 6.6.1. Device Preparation for Windows Devices | 90 |
| 6.6.2. Setting Windows zProperties | 91 |

| | |
|---|-----|
| 6.6.3. Testing WMI on a Windows Server | 91 |
| 6.6.4. Optional Windows Configuration | 91 |
| 6.6.5. Modeling Services on Windows Devices | 92 |
| 6.6.6. Collecting Windows Eventlog Events | 92 |
| 6.6.7. Monitoring Windows Performance with SNMP Informant | 92 |
| 6.6.8. Running winexe Commands on Windows Servers | 92 |
| 7. Event Management | 94 |
| 7.1. About Events | 94 |
| 7.1.1. Basic Event Fields | 94 |
| 7.1.1.1. device and ipAddress Fields | 94 |
| 7.1.1.2. eventState Field | 94 |
| 7.1.1.3. severity Field | 95 |
| 7.1.1.4. summary and message Fields | 95 |
| 7.1.1.5. evid | 95 |
| 7.1.2. Other Fields | 95 |
| 7.1.3. Details | 96 |
| 7.1.4. De-Duplication | 96 |
| 7.1.5. Auto-Clear Correlation | 97 |
| 7.1.6. Event Consoles | 98 |
| 7.1.6.1. Sorting and Filtering Events | 99 |
| 7.1.6.2. Saving a Custom View | 100 |
| 7.1.6.3. Refreshing the View | 100 |
| 7.1.6.4. Viewing Event Details | 101 |
| 7.1.6.5. Selecting Events | 101 |
| 7.1.6.6. Acknowledging Events | 101 |
| 7.1.6.7. Returning Events to New Status | 102 |
| 7.1.6.8. Classifying Events | 102 |
| 7.1.6.9. Exporting Event Data | 102 |
| 7.1.6.10. Moving Events to History (Close) | 102 |
| 7.1.6.11. Returning Events to Active Status | 102 |
| 7.1.6.12. Creating Events | 103 |
| 7.1.7. Event Sources | 103 |
| 7.1.7.1. Generated Events | 103 |
| 7.1.7.2. Captured Events | 103 |
| 7.1.8. Creating Events Manually | 103 |
| 7.1.8.1. Creating Events through the User Interface | 103 |
| 7.1.8.2. Creating Events from the Command Line | 104 |
| 7.1.9. Event Classes | 104 |
| 7.1.9.1. Event Class zProperties | 105 |
| 7.1.10. Mapping and Transformation | 105 |
| 7.1.10.1. Event Class Mappings | 106 |
| 7.1.10.2. Event Class Transform | 107 |
| 7.1.11. Event Life Cycle | 108 |
| 7.1.11.1. Automatic Event Aging | 108 |
| 7.1.11.2. Automatic Historical Event Cleanup | 109 |
| 7.1.12. Event Commands | 109 |
| 7.1.12.1. Creating Event Commands | 109 |
| 7.1.13. Capturing Email Messages as Zenoss Events | 109 |
| 7.1.13.1. ZenMail | 110 |
| 7.1.13.2. ZenPop | 110 |
| 7.1.13.3. Translating Message Elements to the Event | 110 |
| 8. Production States and Maintenance Windows | 111 |
| 8.1. About Production States and Maintenance Windows | 111 |
| 8.2. Production States | 111 |
| 8.2.1. Defining Production States for Devices | 111 |
| 8.3. Maintenance Windows | 111 |
| 8.3.1. Maintenance Window Events | 112 |
| 8.3.2. Creating and Using Maintenance Windows | 112 |

| | |
|---|-----|
| 9. Organizers and Path Navigation | 114 |
| 9.1. About Organizers and Path Navigation | 114 |
| 9.2. Classes | 114 |
| 9.2.1. Viewing Device Classes | 114 |
| 9.2.2. Setting zProperties at the Class Level | 115 |
| 9.2.3. Defining and Applying Templates at the Class Level | 116 |
| 9.2.4. Creating Classes | 117 |
| 9.3. Systems | 117 |
| 9.3.1. Adding, Moving and Nesting Systems | 118 |
| 9.3.1.1. Moving the Sub-System | 119 |
| 9.4. Groups | 119 |
| 9.4.1. Adding Groups | 119 |
| 9.4.1.1. Moving Groups | 119 |
| 9.5. Locations | 120 |
| 9.5.1. Integration with Google Maps | 120 |
| 9.5.1.1. Overview | 120 |
| 9.5.1.2. API Key | 120 |
| 9.5.1.3. Setting an Address for a Location | 120 |
| 9.5.1.4. Clearing the Google Maps Cache | 121 |
| 9.5.1.5. Network Links | 121 |
| 9.5.1.6. Google Maps Example | 121 |
| 9.5.2. Adding, Moving, and Nesting Locations | 122 |
| 9.5.2.1. Moving Sub-locations | 122 |
| 9.6. Inheritance | 122 |
| 10. User Commands | 124 |
| 10.1. About User Commands | 124 |
| 10.2. Defining User Commands | 124 |
| 10.2.1. User Command Example: Echo Command | 125 |
| 11. Managing Users | 126 |
| 11.1. About Zenoss User Accounts | 126 |
| 11.2. Creating User Accounts | 126 |
| 11.3. Editing User Accounts | 127 |
| 11.3.1. Associating Objects in Zenoss with Specific Users | 128 |
| 11.4. User Groups | 130 |
| 11.5. Roles | 131 |
| 11.6. Device Access Control Lists | 132 |
| 11.6.1. About Device Access Control Lists in Zenoss | 132 |
| 11.6.2. Key Elements | 132 |
| 11.6.2.1. Permissions and Roles | 132 |
| 11.6.2.2. Administered Objects | 132 |
| 11.6.2.3. Users and Groups | 132 |
| 11.6.2.4. Assigning Administered Object Access | 132 |
| 11.6.2.5. Portlet Access Control | 133 |
| 11.6.3. Setup and Configuration Examples | 133 |
| 11.6.3.1. Restricted User with ZenUser Role | 133 |
| 11.6.3.2. Restricted User with ZenManager Role | 133 |
| 11.6.3.3. Adding Device Organizers | 133 |
| 11.6.3.4. Restricted User Organizer Management | 133 |
| 11.6.3.5. Viewing Events | 133 |
| 11.6.4. Detailed Restricted Screen Functionality | 133 |
| 11.6.4.1. Dashboard | 133 |
| 11.6.4.2. Device List | 134 |
| 11.6.4.3. Device Organizers | 134 |
| 11.6.4.4. Reporting | 134 |
| 12. Reporting | 135 |
| 12.1. About Zenoss Reporting | 135 |
| 12.2. Organizing Reports | 135 |
| 12.3. Navigating and Sorting Report Results | 135 |

| | |
|--|-----|
| 12.4. Exporting Reports | 136 |
| 12.4.1. Advanced: Add An Export Button to a Report | 136 |
| 12.5. Reports Included With Zenoss | 136 |
| 12.5.1. Device Reports | 136 |
| 12.5.2. Event Reports | 137 |
| 12.5.3. Performance Reports | 137 |
| 12.5.4. User Reports | 139 |
| 12.6. Graph Reports | 139 |
| 12.6.1. Creating a Graph Report | 140 |
| 12.6.2. Adding Graphs | 140 |
| 12.6.3. Customizing Graph Text | 141 |
| 12.6.4. Organizing Graphs | 142 |
| 12.7. MultiGraph Reports | 142 |
| 12.7.1. Creating A MultiGraph Report | 143 |
| 12.7.2. Collections | 144 |
| 12.7.3. Graph Definitions | 145 |
| 12.7.4. Graph Groups | 146 |
| 12.7.5. Graph Order | 147 |
| 12.8. Creating Custom Reports | 147 |
| 12.8.1. Creating Custom Reports Using the ZMI | 147 |
| 12.8.2. Create A Custom Device Report: Example | 147 |
| 12.9. Using Reports to Help Troubleshoot Zenoss Daemons | 148 |
| 12.10. Scheduling Reports | 149 |
| 12.11. Advanced Zenoss Reports | 150 |
| 13. ZenPacks | 151 |
| 13.1. About ZenPacks | 151 |
| 13.1.1. ZenPacks Provided by Zenoss | 151 |
| 13.2. Installing ZenPacks | 151 |
| 13.2.1. Installing from the Command Line | 151 |
| 13.2.2. Installing from the User Interface | 151 |
| 13.2.3. Installing All Core ZenPacks via RPM | 152 |
| 13.3. Creating ZenPacks | 152 |
| 13.3.1. Packaging and Distributing Your ZenPack | 153 |
| 13.4. Removing ZenPacks | 153 |
| 14. General Administration and Settings | 154 |
| 14.1. Email and Pager Settings | 154 |
| 14.1.1. Setting SMTP and SNPP Information | 154 |
| 14.2. Event Manager Settings | 156 |
| 14.2.1. Accessing Event Manager Settings | 156 |
| 14.2.2. Changing Event Database Connection Information | 156 |
| 14.2.3. Changing Event Manager Cache Settings | 157 |
| 14.2.4. Changing Event Manager Maintenance Settings | 157 |
| 14.3. Setting Portlet Permissions | 157 |
| 14.3.1. User Role to ACL Mapping | 157 |
| 14.3.2. Setting Permissions | 157 |
| 14.3.3. Troubleshooting: Users Cannot See All Portlets | 158 |
| 14.4. Backup and Recovery | 158 |
| 14.4.1. Backup (zenbackup) | 159 |
| 14.4.1.1. Backup Options | 159 |
| 14.4.1.2. Backups Tab | 160 |
| 14.4.1.3. Remote Backups | 160 |
| 14.4.2. Restore (zenrestore) | 160 |
| 14.4.2.1. Before You Restore (for Versions Earlier Than 2.4.5) | 160 |
| 14.4.2.2. Restore Options | 160 |
| 14.5. Working with the Job Manager | 161 |
| 14.5.1. Viewing Jobs | 161 |
| 14.5.2. Running the zenjobs Daemon | 161 |
| 14.6. Maintenance and Performance Tuning | 162 |

| | |
|---|-----|
| 14.6.1. Pack ZEO Database | 162 |
| 14.6.2. Log Rotate Script | 162 |
| 14.6.2.1. Zenoss 2.4.x | 162 |
| 14.6.2.2. Zenoss 2.3.3 and Earlier | 162 |
| A. Zenoss Daemon Commands and Options | 164 |
| A.1. Automated Modeling Daemons | 164 |
| A.2. Availability Monitoring Daemons | 164 |
| A.3. Event Collection Daemons | 164 |
| A.4. Performance Monitoring Daemons | 165 |
| A.5. Automated Response Daemons | 165 |
| B. SNMP Device Preparation | 166 |
| B.1. Net-SNMP | 166 |
| B.2. SNMP V3 Support | 166 |
| B.3. Community Information | 167 |
| B.4. System Contact Information | 167 |
| B.5. Extra Information | 167 |
| C. Using an Existing MySQL Server to Store Events | 168 |
| C.1. About | 168 |
| C.2. Procedure | 168 |
| D. Syslog Device Preparation | 169 |
| D.1. Forwarding Syslog Messages from UNIX/Linux Devices | 169 |
| D.2. Forwarding Syslog Messages from a Cisco IOS Router | 169 |
| D.2.1. Other Cisco Syslog Configurations | 169 |
| D.3. Forwarding Syslog Messages from a Cisco CatOS Switch | 170 |
| D.4. Forwarding Syslog Messages using Syslog-ng | 170 |
| E. TALES Expressions | 171 |
| E.1. About Tales Expressions | 171 |
| E.1.1. Examples | 171 |
| E.2. TALES Device Attributes | 171 |
| E.3. Tales Event Attributes | 172 |
| Glossary | 174 |

Chapter 1. About Zenoss

Zenoss is today's premier, open source IT management solution. Through a single, Web-based console, Zenoss enables you to manage the status and health of your infrastructure.

The power of Zenoss starts with its in-depth Inventory and IT Configuration Database. Zenoss creates this database by discovering *managed resources* -- servers, networks, and other devices -- in your IT environment. The resulting configuration model provides a complete inventory of your servers, network devices, and software applications, down to the level of *resource components* (interfaces, services and processes, and installed software).

Once Zenoss discovers the IT infrastructure, it automatically begins monitoring the performance of each device. Zenoss also provides events and fault management features that tie into the configuration database. These features help drive operational efficiency and productivity by automating many of the notification, alerts, escalation, and remediation tasks you perform each day.

1.1. Zenoss High-Level View

Using agent-less technology, Zenoss monitors your entire IT infrastructure stack, including network, servers, HVAC and power, and even applications. At its highest level, Zenoss comprises these major areas:

- Discovery and configuration
- Performance and availability
- Fault and event management
- Alerting and remediation
- Reporting

Zenoss unifies these areas into a single system with a modern, interactive Web user interface.

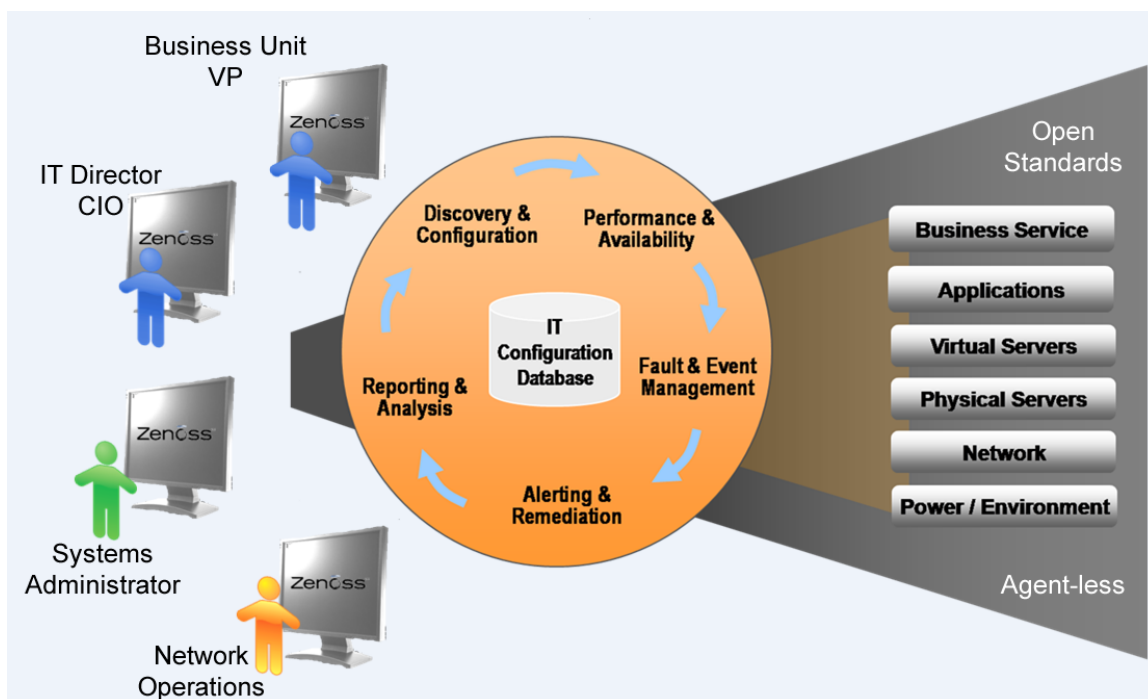


Figure 1.1. Zenoss High-Level View

1.1.1. Key Tenets

Zenoss was designed with these important ideas at its core:

- **Modeling**

Zenoss' model enables it to understand the environment in which it operates. Through sophisticated and detailed analysis, Zenoss determines how to monitor and manage complex IT environments. The core of the Zenoss standard model describes basic information about each device's operating system and hardware. The model is object-based, and is easily extended through object inheritance.

- **Discovery**

With a sophisticated model, manual input and maintenance of data is challenging. To address this challenge, Zenoss uses *discovery* to populate the model. During discovery, Zenoss accesses each monitored device in your infrastructure and interrogates it in detail, acquiring information about its components, network integration, and dependencies.

- **Normalization**

Because Zenoss collects information from different platforms and through different protocols, the amount and format of available information varies. For example, file system information gathered from a Linux server differs from similar information gathered from a Windows server. Zenoss standardizes the data gathered so that you can perform valid comparisons of metrics gathered by different methods and for different systems.

- **Agentless Data Collection**

To gather information, Zenoss relies on agent-less data collection. By communicating with a device through one of several protocols (including SNMP, SSH, Telnet, and WMI), it minimizes the impact on monitored systems.

- **Full IT Infrastructure**

Unlike other tools, Zenoss' inclusive approach unifies all areas of the IT infrastructure--network, servers, and applications--to eliminate your need to access multiple tools.

- **Configuration Inheritance**

Zenoss extends the concept of inheritance in object-oriented languages to configuration. All core configuration parameters (known as *zProperties*) and monitoring directions (*monitoring templates*) use inheritance to describe how a device should be monitored. Inheritance allows you to describe, at a high level, how devices should be monitored. It also supports ongoing refinements to the configuration. (For detailed information on inheritance and templates, refer to the chapter titled "Properties and Templates.")

- **Cross-Platform Monitoring**

Zenoss monitors the performance and availability of heterogeneous operating systems (including Windows, Linux, and Unix), SNMP-enabled network devices (such as Cisco), and a variety of software applications (such as WebLogic and VMware).

- **Scale**

You can deploy Zenoss on a single server to manage hundreds of devices. Zenoss Enterprise allows you to manage large, distributed systems by using horizontal scaling of its collectors.

- **Extensibility**

Zenoss' extension mechanism, ZenPacks, allow for rapid addition and modification to customize your environment.

1.2. Zenoss Architecture and Technologies

The following diagram illustrates Zenoss' system architecture.

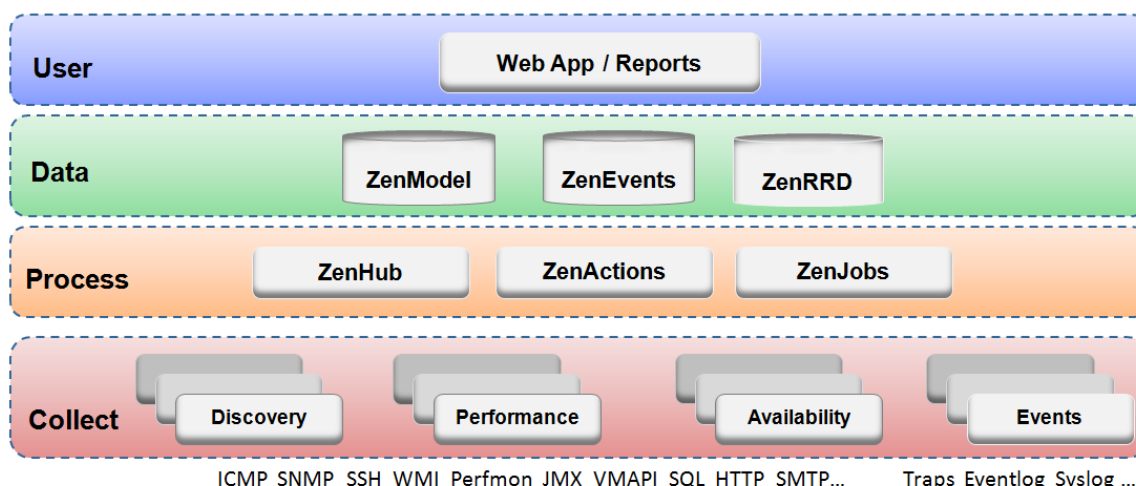


Figure 1.2. Zenoss Architecture

Zenoss is a tiered system with four major parts:

- User layer
- Data layer
- Processing layer
- Collection layer

1.2.1. User Layer

Built around the Zope Web application environment, the user layer is manifested as a Web portal. It uses several JavaScript libraries, Mochi Kit, YUI, and extJS to provide a rich application experience.

Through the user interface, you access and manage key Zenoss components and features. From here, you can:

- Watch the status of your enterprise, using the Zenoss Dashboard
- Work with devices, networks, and systems
- Monitor and respond to events
- Manage users
- Create and run reports

The user layer Interacts with the data layer and translates the information for display in the user interface.

1.2.2. Data Layer

Zenoss configuration and collection information is stored in the data layer, in three separate databases:

- **ZenRRD** - Utilizing RRDtool, stores time-series performance data. Because RRD files are stored locally to each collector, no bottlenecks result from writing to a single database as new collectors are added.
- **ZenModel** - Serves as the core configuration model, which comprises devices, their components, groups, and locations. It holds device data in the ZEO back-end object database.
- **ZenEvents** - Stores event data in a MySQL database.

1.2.3. Process Layer

The process layer manages communications between the collection and data layers. It also runs back-end, periodic jobs, as well as jobs initiated by the user (ZenActions and ZenJobs). The process layer utilizes Twisted PB (a bi-directional RPC system) for communications.

1.2.4. Collection Layer

Zenoss' collection layer comprises services that collect and feed data to the data layer. These services are provided by numerous daemons that perform modeling, monitoring, and event management functions.

The modeling system uses SNMP, SSH, and WMI to collect information from remote machines. The raw information is fed into a plugin system (*modeling plugins*) that normalizes the data into a format that matches the core Zenoss model.

Monitoring daemons track the availability and performance of the IT infrastructure. Using multiple protocols, they store performance information locally in RRD files, thus allowing the collectors to be spread out among many collector machines. Status and availability information, such as ping failures and threshold breaches, are returned through ZenHub to the event system.

For more information about Zenoss daemons, see the appendix titled "Zenoss Daemon Commands and Options."

1.3. Zenoss' Monitoring Approach

Zenoss uses a model-driven approach to monitoring, combining discovery and the model to enable automatic monitoring. This strategy reduces system maintenance overhead and ensures that new devices and applications are monitored as they come online.

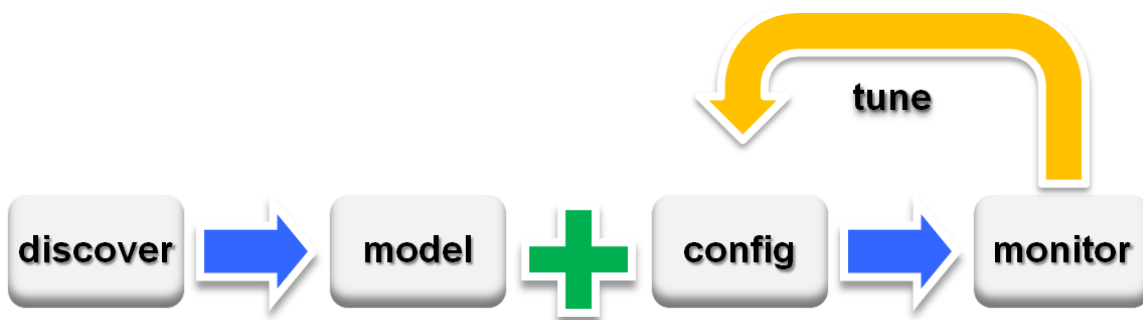


Figure 1.3. Workflow: Model-Driven Monitoring

As shown in the previous illustration, model-driven monitoring begins with discovery, which populates the model. It continues as the configuration defined in the model is automatically applied and monitoring begins. As the system runs, the configuration is further fine-tuned.

Zenoss model-driven monitoring approach is demonstrated by the following file system monitoring scenario.

1.3.1. File System Monitoring

By default, Zenoss is configured with a file system threshold of 90% utilization. Each time Zenoss discovers a file system, this threshold is automatically applied to the file system, and monitoring begins.

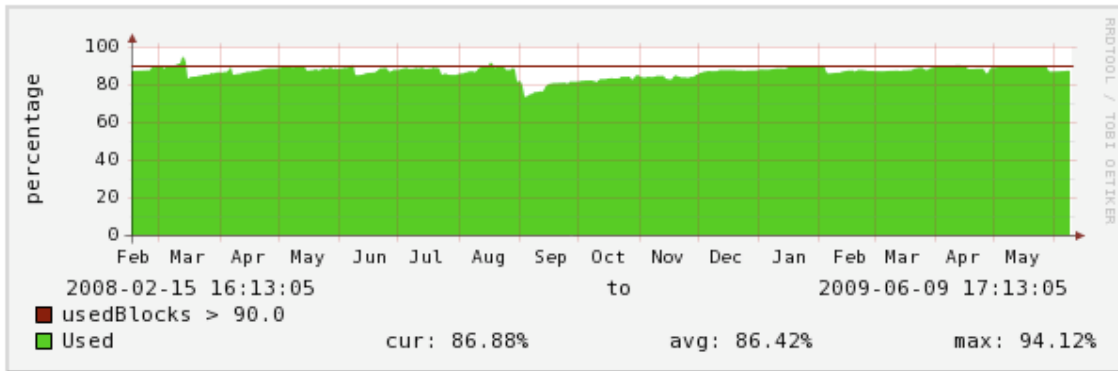


Figure 1.4. Monitored File System (Threshold Exceeded)

This illustration shows the result of a system being monitored, using the default configuration. The graph shows that the threshold of 90% has been exceeded numerous times. Because the data in the model is normalized, thresholds will apply regardless of the collection mechanism (SNMP, SSH, and WMI).

The chapter titled "Properties and Templates" provides more information about modifying Zenoss' monitoring configuration.

1.4. Zenoss Terminology

You should understand Zenoss' use of the following terms when working with the system.

Glossary

| | |
|------------------|--|
| alert | Email or page sent as a result of an event. |
| data point | Data returned from a data source. In many cases, there is only one data point for a data source (such as in SNMP); but there may also be many data points for a data source (such as when a command results in the output of several variables). |
| data source | Method used by Zenoss to collect monitoring information. Example data sources include SNMP OIDs, SSH commands, and perfmon paths. |
| device | Primary monitoring object in Zenoss. Generally, a device is the combination of hardware and an operating system. |
| device class | Special type of organizer used to manage how the system models and monitors devices (through zProperties and monitoring templates). |
| device component | Object contained by a device. Components include interfaces, OS processes, file systems, CPUs, and hard drives. |
| discovery | Process by which Zenoss gathers detailed information about devices in the infrastructure. Results of discovery are used to populate the Zenoss model. |
| event | Manifestation of important occurrence within the system. Events are generated internally (such as when a threshold is exceeded) or externally (such as through a syslog message or SNMP trap). |
| event class | Categorization system used to organize event rules. |
| event rules | Controls how events are manipulated as they enter the system (for example, changing the severity of an event). zProperties configure event rules. |

| | |
|---------------------|---|
| graph | Displays one or more data points, thresholds, or both. |
| managed resource | Servers, networks, virtual machines, and other devices in the IT environment. |
| model | Representation of the IT infrastructure in Zenoss. The model tells Zenoss "what is out there" and how to monitor it. |
| monitoring template | Description of what to monitor on a device or device component. Monitoring templates comprise four main elements: data sources, data points, thresholds, and graphs. |
| organizer | Hierarchical system used to describe locations and groups. Zenoss also includes special organizers, which are classes that control system configuration. |
| resource component | Interfaces, services and processes, and installed software in the IT environment. |
| threshold | Defines a value beyond which a data point should not go. When a threshold is reached, Zenoss generates an event. Typically, threshold events use the / Perf event class. |
| zProperty | Configuration property defined on a device or event class. zProperties control a large part of how monitoring is performed in Zenoss. Configuration of zProperties relies on inheritance. |

Chapter 2. Using Zenoss

Read the following sections to learn more about working in the Zenoss interface, and to learn how to:

- Customize the dashboard
- Search for devices
- Navigate the event console
- Run Zenoss commands
- Create and use alerts
- Create custom event views

2.1. Zenoss Interface and Navigation

After you install Zenoss and navigate to the interface from your Web browser, the Zenoss Dashboard appears. The Zenoss Dashboard provides at-a-glance information about the status of your IT infrastructure. It is the primary window into devices and events that Zenoss enables you to monitor.



Figure 2.1. Zenoss Dashboard

The Dashboard can show:

- Zenoss information resources and Web pages
- Important error-level device events
- Geographical high-level view
- "Troubled" devices

Key Dashboard and interface areas include:

- Navigation menu
- Breadcrumbs

- User information area
- Portlets
- Zenoss Network Map

2.1.1. Navigation Menu

The Navigation menu lets you access most of the Zenoss' features. The menu is divided among several functional areas:

- **Main Views**, which includes these selections:
 - **Dashboard** - Returns you to the primary view.
 - **Event Console** - Lists all current events in the event database.
 - **Device List** - Shows a list of all devices in Zenoss.
 - **Network Map** - Shows a graphical representation of the devices in your network.
- **Classes**, which includes these selections:
 - **Events** - Links to the event management area, where you can monitor event status, events, history, zProperties, and event transforms. You also can track changes made to events.
 - **Devices** - Lets you manage sub-devices and a summary of events by severity. Allows you to view events sorted by severity, followed by device name, history of device events, PerfConfig, zProperties for devices, and recent device changes.
 - **Services** - Allows you to show service classes, administer commands on a service basis, access zProperties, and track changes made to services monitoring.
 - **Processes** - Lets you create new process groupings and add processes to monitor.
 - **Products** - Shows a list of all manufacturers of devices in the Zenoss database.
- **Browse By**, which lets you see data based on any of the local groupings Zenoss enables you to create. Selections include:
 - **Systems** - Lets you see network status, categorized into the system groupings you create.
 - **Groups** - Provides access to the same data as when browsing by Systems, with the exception of performance data.
 - **Locations** - Allows you to see data related to devices grouped by physical locations.
 - **Networks** - Shows devices and sub-networks, based on IP address groupings.
 - **Reports** - Lets you view and define reports in Zenoss.
- **Management**, which includes:
 - **Add Device** - Add devices to Zenoss.
 - **MIBs** - Add and manage SNMP MIBs in Zenoss.
 - **Collectors** - Add collectors to Zenoss for better performance and scaling when the device count is too large for one collector.
 - **Settings** - Manage other Zenoss settings, such as dashboard production state, state conversions, and administrative roles.
 - **Event Manager** - Lets you view and manage back-end event management configuration.

The following figure illustrates key selections from the Navigation menu.

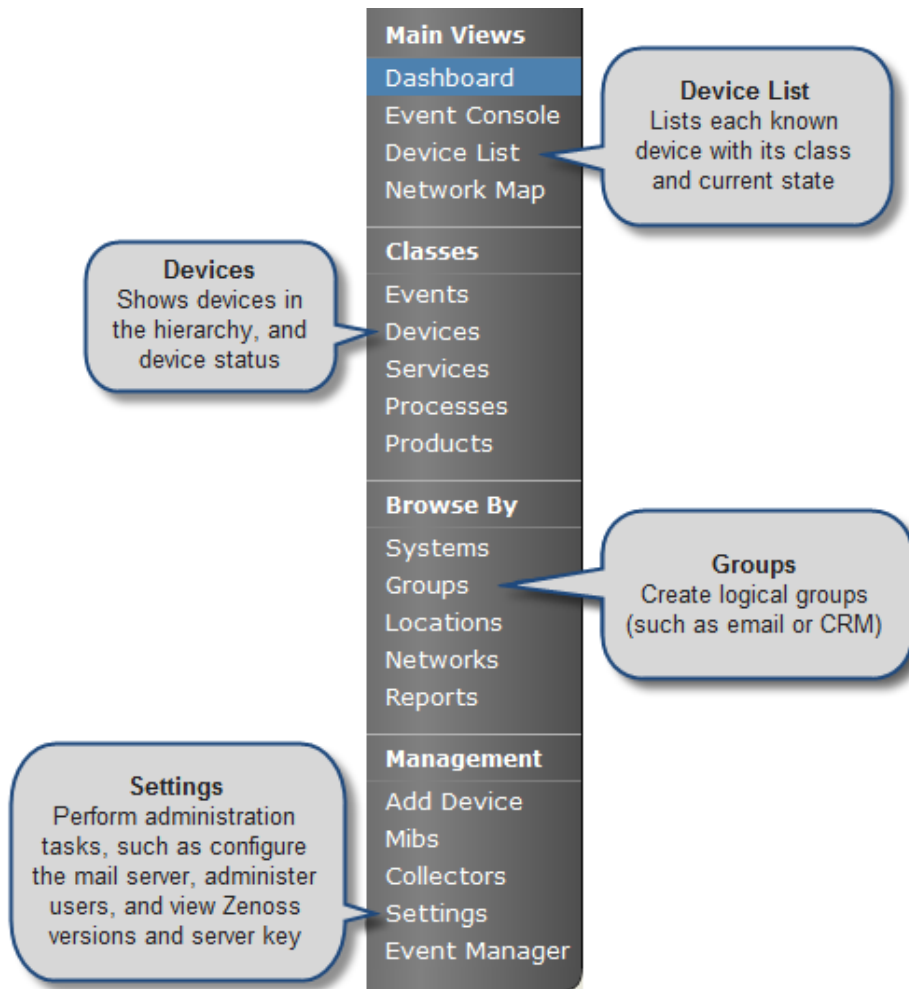


Figure 2.2. Navigation Menu

2.1.1.1. Hiding the Menu

Click the triangular indicator at the top of the Navigation menu to hide or display menu selections.

2.1.1.2. Pinning the Menu

Click the pin icon to "pin" the menu into place, keeping it visible in all views.

2.1.2. Breadcrumbs

The breadcrumbs area shows your current location in Zenoss. Use this trail to keep track of your location and navigate to previously selected pages in the interface hierarchy.

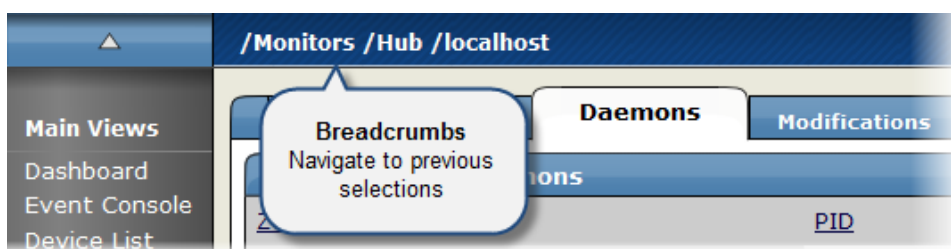


Figure 2.3. Breadcrumbs (Navigation)

2.1.3. User Information Area

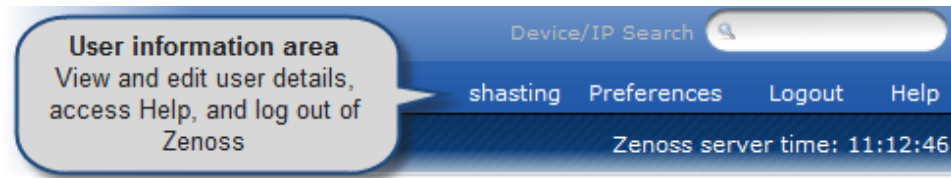



Figure 2.4. User Information Area

The User information area offers information and selections:

- **Login ID** - The ID of the user currently logged in to Zenoss appears at the far left of this area.
- **Preferences** - Click to edit user settings, such as authentication information, roles, and groups. (You also can access user settings from the Navigation menu Settings selection.)

 From other Preferences tabs, you can manage administered objects, event views, and alerting rules.

- **Logout** - Click to log out of Zenoss.
- **Help** - Click to access Zenoss community product documentation, FAQs, and HowTos, at:

<http://community.zenoss.org/community/documentation>

2.1.3.1. Update Details

The date and time that Zenoss was last updated appears below the Preferences link. Every 60 seconds, Zenoss polls for new data and refreshes the data fields. If the poll fails, then Zenoss displays the message:

Lost Connection to Zenoss

2.1.4. Portlets

The main content of the Dashboard comprises portlets, which provide information about the system and your infrastructure. Portlets that you can display on the dashboard are:

- **Site Window** - Initially provides links to Zenoss resources, including product guides, forums, and training events. (The URL for the default content is <http://www2.zenoss.com/in-app-welcome>.) You can customize this portlet to display content from any URL.
- **Device Issues** - Displays a list of devices, associated with color-coded events of error or critical severity levels. Click a device in the list to view its event log.







| Device | Events |
|---|--------|
|  localhost | 1 |
|  marc-irlandezs-computer.loc | 2 |
|  hp3055 | 2 |
|  hp3055 | 2 |
|  marc-irlandezs-computer.loc | 2 |
|  cent5_java | |

Figure 2.5. Device Issues Portlet

- **Google Maps** (device locations) - Shows configured locations and configured network connections.



Figure 2.6. Google Maps Portlet

- **Zenoss Issues** - Contains system self-monitoring information.
- **Production States** - Shows devices assigned to a particular production state.
- **Top Level (Root) Organizers** - Lists status for each grouping in your defined Zenoss hierarchy.

| Root Organizers | |
|---------------------|--------|
| Object | Events |
| /Devices/Server | 1 |
| /Devices/Discovered | 4 |
| /Devices/Network | |
| /Devices/Printer | |
| /Devices/Power | |
| /Devices/KVM | |

Figure 2.7. Top Level Organizers Portlet

- **Object Watch List** - Allows the display of high-level status device classes, groups, systems, event classes, and locations that you select.

2.1.4.1. Customizing Portlets

You can customize each portlet that appears on the Dashboard. Customization options vary depending on the portlet type.

Click * (asterisk), which appears at the top right corner of a portlet, to view and customize display options. Click **Save Settings** to save your selections and then return to main portlet content.

The following table lists information you can customize for each Zenoss portlet.

| For this portlet type.. | ...you can customize: |
|-----------------------------|---|
| Welcome | Title, Refresh Rate, Destination URL |
| Device Issues | Title, Refresh Rate |
| Google Maps | Title, Refresh Rate, Base Location |
| Zenoss Issues | Title, Refresh Rate |
| Production States | Title, Refresh Rate, Production States (to appear on the Dashboard) |
| Top Level (Root Organizers) | Title, Refresh Rate, Root Organizer (to appear on the Dashboard) |

2.1.4.2. Adding and Duplicating Portlets

To add a portlet, select Add portlet (located below the Zenoss server time display at the top right of the Dashboard). From the Add Portlet dialog, you can add a portlet or restore portlets to the default view.

Your Dashboard can display more than one of the same portlet type. You might want to display duplicate portlets, for example, to get at-a-glance information about more than one device location that appears in the Google Maps portlet.

2.1.5. Zenoss Network Map

The Network Map represents your network's Layer 3 topology. From the map, you can quickly determine the status of each device by its background color.

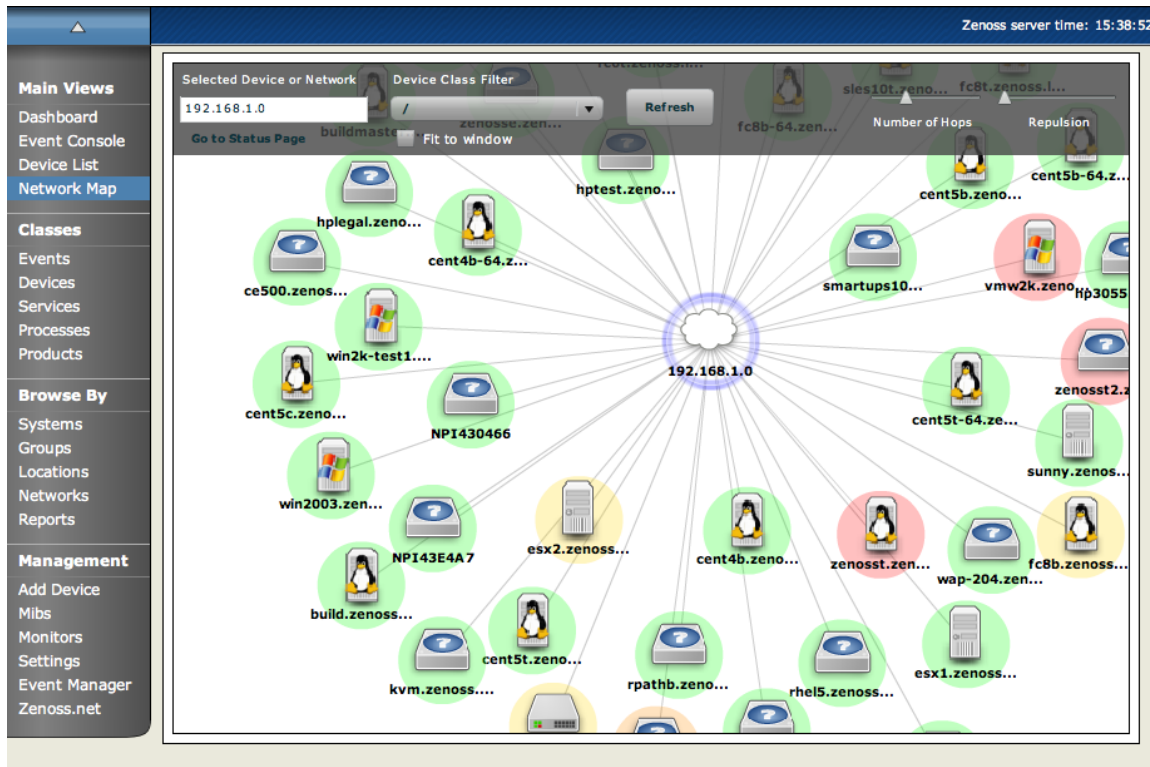


Figure 2.8. Zenoss Network Map

2.1.5.1. Choosing the Network to Display

The network displayed is configured for each user. From the Preferences area, modify Network Map Start Object to indicate a network, and then click **Save**.

2.1.5.2. Viewing Device and Network Details

Double-click a device or network icon in the map to focus on it. Focusing on a node:

- Centers it on the map
- Shows links from the node, based on the number of hops selected

Alternatively, you can type the name or IP address of a device or network in the Selected Device or Network field, and then click **Refresh** to focus on that node.

i When you select a node, the network map displays only the links that are currently loaded into the map. It does not download and display new link data.

2.1.5.3. Loading Link Data

To load link data for a node:

1. Double-click the node on the map to focus on it, or enter the device name or IP address in the Selected Device or Network field.
2. Select the number of hops to download and display.
3. Click **Refresh**.

2.1.5.4. Filtering by Device Type

You can filter the devices that appear on the network map. To do this, select a filter from the Device Class Filter list of options. For example, to show only Linux devices on the map, select /Server/Linux from the list of options, and then click **Refresh**.

2.1.5.5. Adjusting Viewable Hops

You can adjust the number of hops that appear on the network map. Use the Number of Hops slider, which adjusts the number of hops from 1 to 4.

2.1.5.6. Adjusting the Network Map

Use the Repulsion slider to expand or contract the icons that appear on the map. Move the slider right to expand the icons, or left to contract them.

Select the Fit to Window option to bring all displayed icons into the viewable area.

2.1.5.7. Viewing Device or Network Details

To see detailed information about a device or network, select it in the map, and then click **Go to Status Page**.

2.1.6. Zenoss Menus

Zenoss offers two types of menus from which you make selections:

- Page menus
- Table menus

2.1.6.1. Page Menus

Page menus extend the tabs that appear at the top of the page. Generally, actions initiated through a page menu affect the object or objects that the page represents. This could be, for example, a device or any group of devices.

As shown in the following figure, the Page menu is expanded next to the Classes tab.



Figure 2.9. Page Menu

2.1.6.2. Table Menus

Table menus generally affect objects in a table. Access table menus by clicking the triangle next to a table title on a page. As shown in the following figure, the Sub-Devices table menu is expanded.

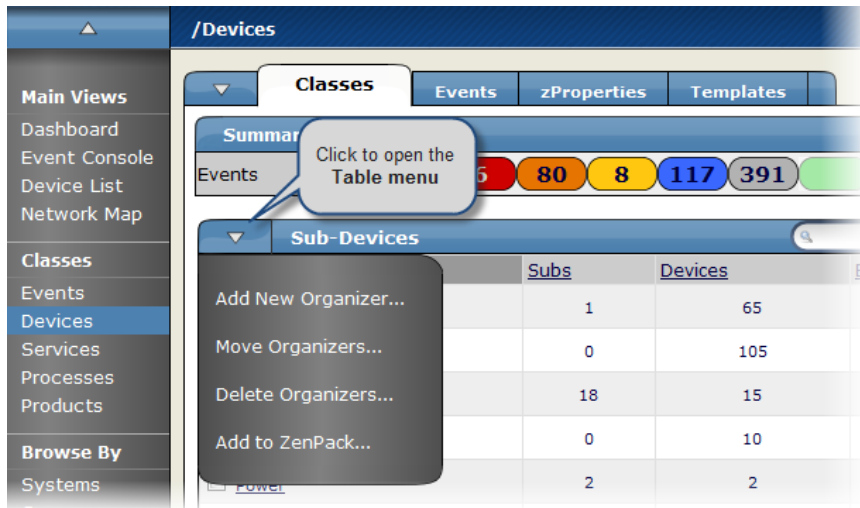


Figure 2.10. Table Menu

2.2. Customizing the Zenoss Dashboard

You can customize the Zenoss Dashboard by:

- Selecting the portlets you want to monitor
- Arranging portlets
- Changing the Dashboard column layout



Figure 2.11. Customize Dashboard

2.2.1. Selecting Portlets

To add a portlet to the Dashboard:

1. Click **Add portlet** (located at the top right of the Dashboard main area).

The Add Portlet dialog appears.

2. Select a portlet.

The portlet appears at the top right of the Dashboard main area.

To remove a portlet from the Dashboard:

1. Click * (asterisk) that appears at the top right corner of the portlet you want to remove.

The portlet expands to show its Settings area.

2. Click **Remove Portlet**.

2.2.2. Arranging Portlets

To arrange portlets on the Dashboard, click the portlet header and drag the portlet to any location on the Dashboard. Other portlets rearrange depending on the location you drop it.

2.2.3. Changing the Dashboard Column Layout

You can change the layout of the Dashboard to one, two, or three-column displays. For two-column display, you can additionally choose a layout that offers columns of equal or varying widths.

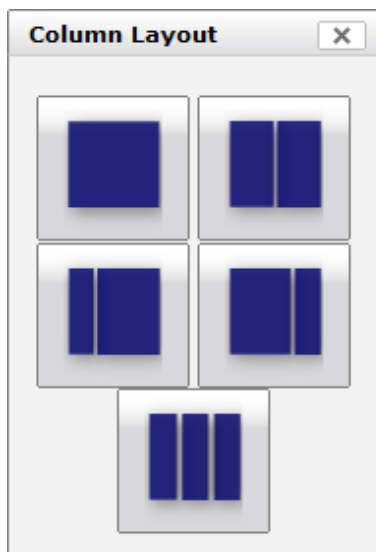



Figure 2.12. Column Layout Dialog

To change the Dashboard column layout:

1. Click Configure layout... (located at the top right of the Dashboard main area).

The Column Layout dialog appears.

2. Click to select your preferred column layout.

 After selecting a new layout, you likely will need to rearrange the portlets on the Dashboard.

2.3. Searching for Devices

To locate a specific device in Zenoss, use one of these facilities:

- **Device/IP Search** - Enter a device name or IP address in this field, located at the top right of the Zenoss interface.

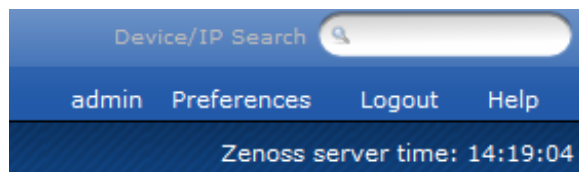


Figure 2.13. Device/IP Search Area

- **Browse By** - Select an option from the Browse By area of the Navigation menu if you do not know the device name, or if you are not searching for a specific device. You can browse by:
 - **Systems** - Browse by common device types, such as file servers, printers, and infrastructure.
 - **Groups** - Browse by groups that you set up to organize your devices.
 - **Locations** - Browse by groups based on location.

2.4. Navigating the Event Console

The event console is Zenoss' central nervous system, enabling you to view and manage events. It displays the repository of all events that are detected by the system.

To access the event console, click Event Console in the Navigation menu.

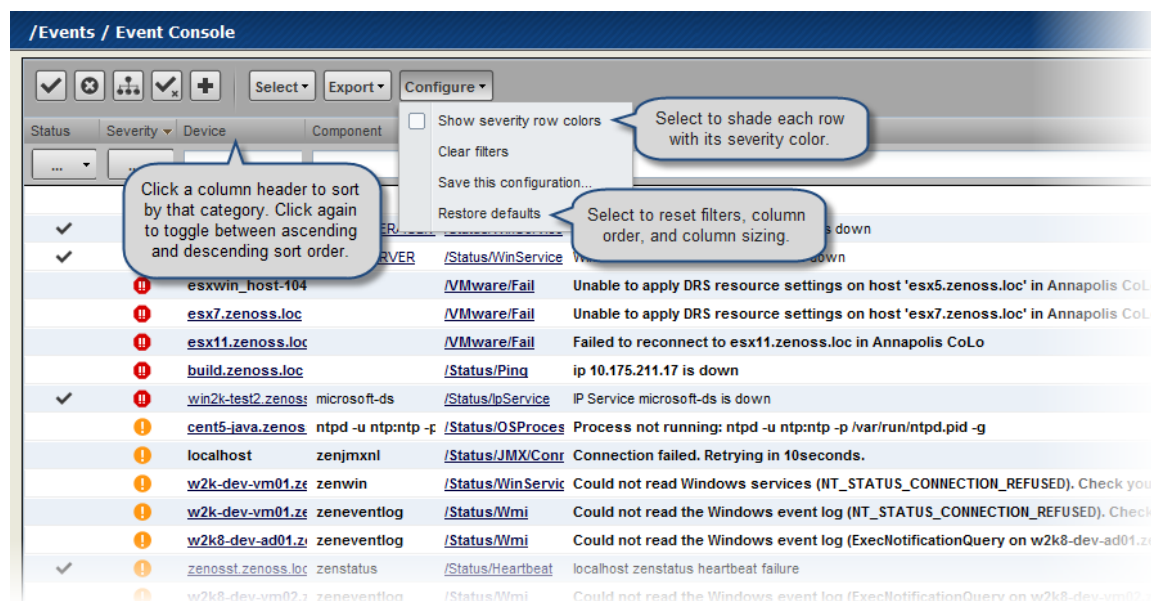


Figure 2.14. Zenoss Event Console

2.4.1. Sorting and Filtering Events

You can sort and filter events that appear in the event console to customize your view.

You can sort events by any column that appears in the event console. To sort events, click a column header. Clicking the header toggles between ascending and descending sort order.

Filter options appear below each column header.

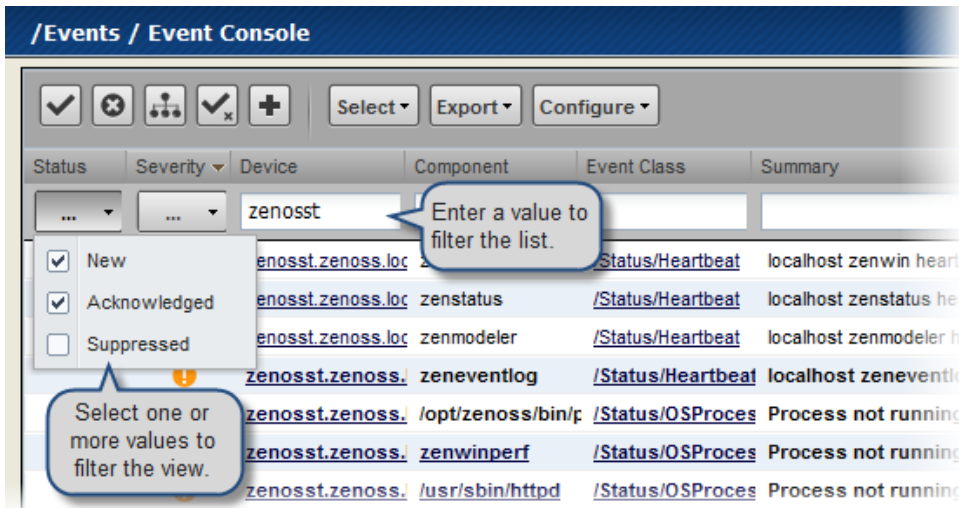


Figure 2.15. Event Console Filter Options

You can filter the events that appear in the list in several ways, depending on the field type. Date fields (such as First Seen and Last Seen) allow you to enter a value or use a date selection tool to limit the list. For other fields, such as Device, Component, and Event Class, enter a match value to limit the list.

The Count field allows you to filter the list when compared to a value:

- n - Displays events with counts greater than or equal to that value.
- $<n$ - Displays events with counts less than that value.
- $\leq n$ - Displays events with counts less than or equal to that value.
- $=n$ - Displays events with counts equal to that value.

To clear filters, select **Configure > Clear filters**.

2.4.1.1. Saving a Custom View

You can save your custom event console view by bookmarking it for quick access later. To do this:

1. Select **Configure > Save this configuration**.

A dialog containing a link to the current view appears.

2. Click and drag the link to the bookmarks link on your browser's menu bar.

Zenoss adds a link titled "Event Console" to your bookmarks list.

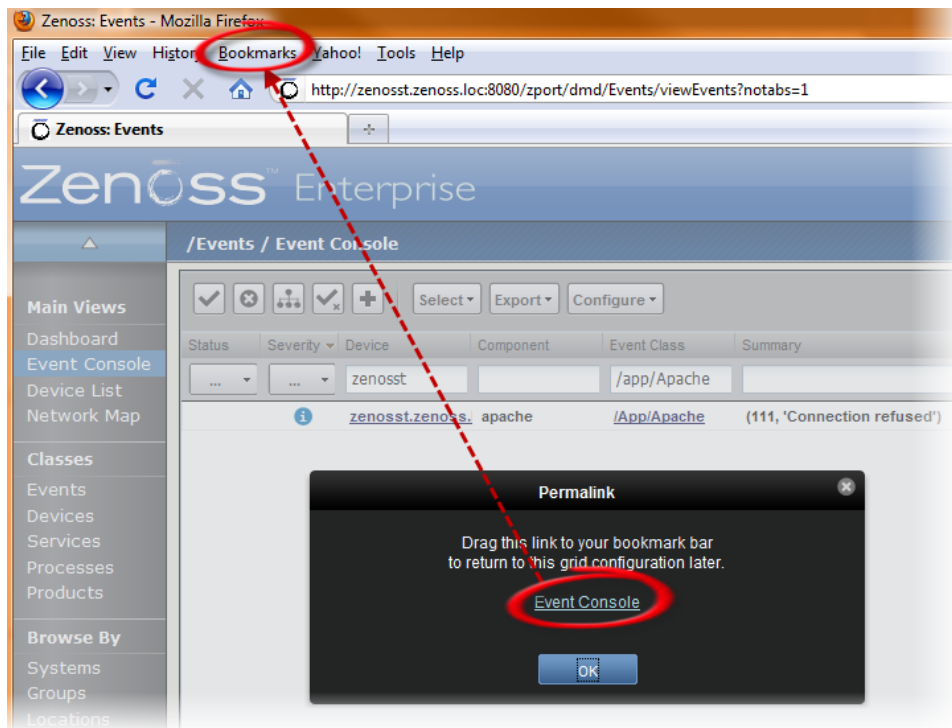


Figure 2.16. Saving a Custom View (Bookmark)

Tip: You may want to re-title the bookmark, particularly if you choose to save more than one event console view.

2.4.2. Refreshing the View

You can refresh the list of events manually or specify that they refresh automatically. To manually refresh the view, click **Refresh**. You can manually refresh at any time, even if you have an automatic refresh increment specified.

To configure automatic refresh, select one of the time increments from the Refresh list. By default, automatic refresh is enabled and set to refresh each minute.

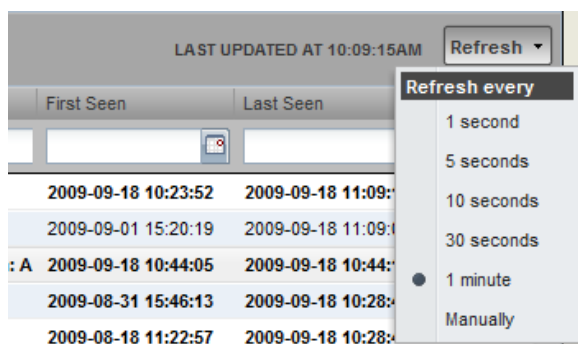


Figure 2.17. Automatic Refresh Selections

2.4.3. Viewing Event Details

You can view details for any event in the system. To view details, double-click an event row.

Tip: Do not double-click on or near the device name, component, or device class in the row. Doing this displays details about that entity, rather than information about the event.

The Event Detail area appears.

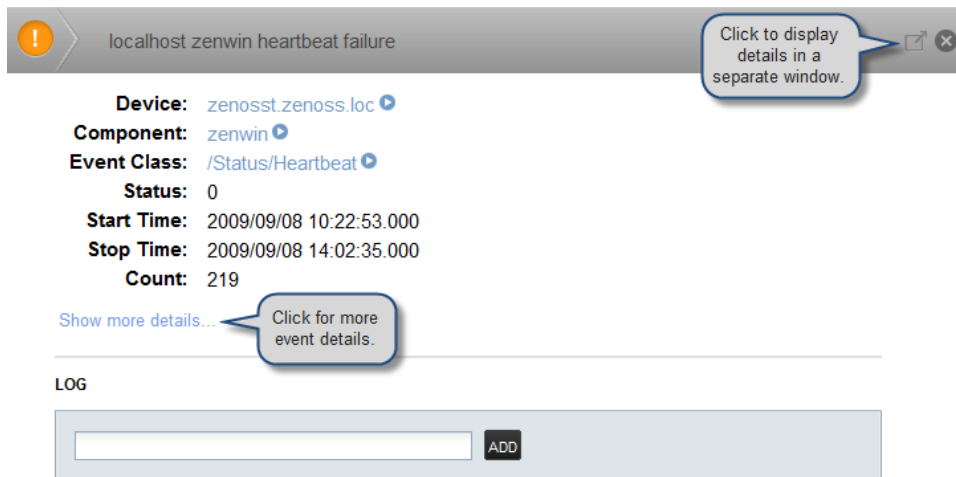


Figure 2.18. Event Detail

To see more information about the event, click Show more details.

You can use the Log area to add specific information about the event. Enter details, and then click **Add**.

2.4.4. Selecting Events

To select one or more events in the list, you can:

- Click a row to select a single event
- Ctrl-Click rows to select multiple events, or Shift-Click to select a range of events
- Click Select to select all, none, new, acknowledged, or suppressed events

2.4.5. Managing Events

You can manage events from the event console. After selecting an event, you can:

- Acknowledge the event
- Close the event (move it to history)
- Map the event, associating it with a specific event class
- Return the event to New status (revoke its Acknowledged status)

You also can add an event from the event console.

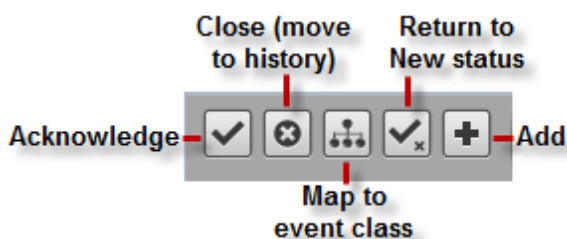


Figure 2.19. Event Management Options

2.5. Running Commands from the User Interface

Zenoss allows commands to be run through the Web-based user interface. You can run commands on a single device or on a group of devices.

Zenoss includes several built-in commands, such as ping and traceroute.

To run commands from the user interface:

1. Navigate to the device or device group where you want to run the command. You must have the command defined here.
2. From the Device page menu, select Run Commands, and then select the command name you want to run.

Zenoss runs the command. Command output appears on the screen.

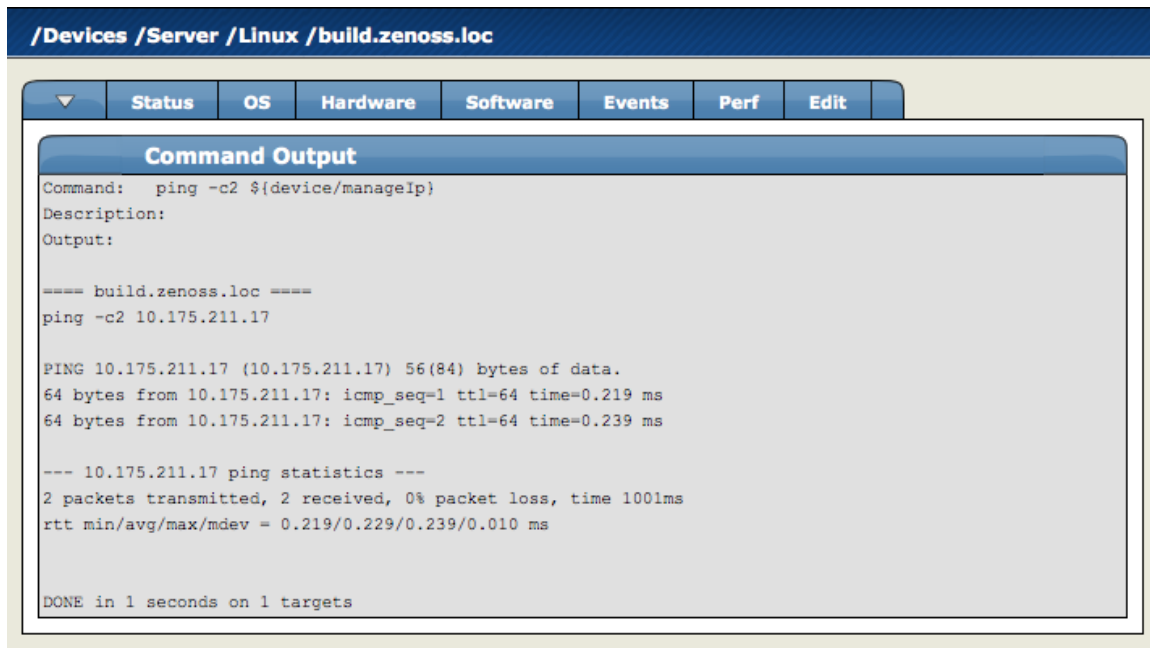


Figure 2.20. Command Output

2.6. Creating and Using Alerts

You can implement *alerts* to send email or pages based on events that Zenoss receives. Implemented by the `zenactions` daemon, Zenoss continuously evaluates each user's paging rules against the event database. Each user has his own set of alerting rules.

Read the following sections to learn about:

- Setting SMTP settings for alerts
- Creating alerting rules
- Escalating alerts
- Scheduling alerts

2.6.1. Setting SMTP Settings For Alerts

To use email and pager alerts, Zenoss must point to an SMTP relay with the proper settings.

1. From the navigation menu, select Settings.

The Settings page appears.

| State at time: 2008/06/18 13:43:14 | |
|--|--|
| SMTP Host | localhost |
| SMTP Port (usually 25) | 25 |
| SMTP Username (blank for none) | docs |
| SMTP Password (blank for none) | |
| From Address for Emails | zenosst@zenoss.com |
| Use TLS? | <input type="checkbox"/> |
| Page Command | \$ZENHOME/bin/zensnpp I |
| Dashboard Production State Threshold | 1000 |
| Dashboard Priority Threshold | 2 |
| State Conversions | Production: 1000 Pre-Production: 500 Test: 400 Maintenance: 300 Decommissioned: -1 |
| Priority Conversions | Highest: 5 High: 4 Normal: 3 Low: 2 Lowest: 1 Trivial: 0 |
| Administrative Roles | Administrator Analyst Engineer Tester |
| Google Maps API Key Help | zenoss |
| Save | |

Figure 2.21. Settings Tab - SMTP Settings

- To set up the mail servers, you must configure the SMTP Host, the SMTP Port, SNPP Host, and the SNPP Port.

Now you are prepared to create and use alerting rules for the Zenoss system.

2.6.2. Creating an Alerting Rule

Alerting rules are created on a per user basis. You can add additional recipients for rules, but upon creation, the rules are tied to a user account.

- From the upper right corner of the Zenoss Dashboard, click the Preferences link.

The Preferences page appears.

Figure 2.22. Preferences - Edit Tab

2. Select the Alerting Rules tab.

The Alerting Rules tab appears.

3. From the Alerting rule table menu, select Add Alerting Rule.

The Add Alerting Rule dialog appears.

Figure 2.23. Add Alerting Rule

4. In the ID field, enter a name for the alert.
5. Click **OK**.

The main Alerting Rules page appears, showing the alert you just created.

6. Click the name of the alert.

The Alert Details page appears.

Figure 2.24. Edit Alert Details

2.6.2.1. Define and Enable the Alert

Set the attributes from the Alert Details page.

1. Use the Delay field to set the number of seconds to wait before sending the alert. If an event clears before delay time no alert is sent.
2. To enable the alert, set Enabled to True.
3. Use the Repeat Time to set the time for repeating the alert to send the alert every x seconds until the event is acknowledged.
4. In the Action field, select whether you want the system to send email or a page.

If action is defined as email the event will be emailed. If the default action is set to page, you must define and test the "Page Command" (from the Settings > Settings tab). Many wireless phone systems have SMTP to Simple Messaging Service (SMS) gateways, so in some cases, you also can use email to send pages.

By default, email alerts are sent to the email address for this user. Pager alerts go to the specified pager address. You can override this by filling in the Address (optional) field.

5. The Where area of the tab sets the thresholds for the Alert.

The default rule that is created contains the thresholds for an event occurrence where the Event State is "New," Severity is "greater than Error," and Production State is "Production." You can change these thresholds by changing the values in the pop-up menus.

6. You also can add more filters to the Where area by choosing a filter from the Add Filter menu. Adding a filter creates a pop-menu in the Where area from which you can choose additional values to filter the event. To Remove any of the filters for the alert, click the (-) minus button.
7. Click **Save** to save the values you entered on this tab.

Notes:

- Setting Enabled to True disables all alert windows, and is the same as a 24x7 alerting window.
- To alert only during certain periods specified in the alerting windows, set Enabled to False.
- To ensure that an alerting rule will not send alerts, ensure that Enabled is set to False and that all alerting rule windows are not enabled.

2.6.2.2. Create the Content of the Alert Message

1. From the Alerting Rules Page, click the Message tab to customize the message that is sent to the specified address.

The Message tab appears.

/ZenUsers /docs /Alerting Rules /AB-test Zenoss server time: 8:03:03

Message [Edit] [Schedule]

State at time: 2008/07/17 08:02:47

Message (or Subject)

[zenoss] %(device)s %(summary)s

Body

Device: %(device)s
 Component: %(component)s
 Severity: %(severityString)s
 Time: %(firstTime)s
 Message:
 %(message)s
 Event Detail

Clear Message (or Subject)

[zenoss] CLEAR: %(device)s %(clearOrEventSummary)s

Clear Body

Event: '%(summary)s'
 Cleared by: '%(clearSummary)s'
 At: %(clearFirstTime)s
 Device: %(device)s
 Component: %(component)s
 Severity: %(severityString)s
 Message:

Save

Message Format is a python format string. Fields are specified as %(fieldname)s. The list of fields available in the event database is: dedupid, evid, device, component, eventClass, eventKey, summary, message, severity, eventState, eventClassKey, eventGroup, stateChange, firstTime, lastTime, count, prodState, suppid, manager, agent, DeviceClass, Location, Systems, DeviceGroups, ipAddress, facility, priority, ntevid, ownerid, clearid, DevicePriority, eventClassMapping, monitor, iprealm.

Figure 2.25. Alerting Rules Message Tab

2. Use the Message tab to specify the email message subject and body. You actually have two messages to create. The first (called Message) is the message to send when the thresholds for the alert are met or exceeded. The second message is the one to send when the event has cleared (called Clear Message).

The fields for the subject and message areas are Python format strings.

3. Click Save to save the data you entered on this page.

2.6.2.3. Create a Schedule for Sending the Alert

By default, all enabled schedules are active at all times. If you want to restrict the times for which an alerting rule is active, follow these steps:

1. Set the Enabled alert field to False.
2. From the Alerting Rules page, click the Schedule tab to set up a schedule for the alert.

The Schedule tab appears.

3. To add a new schedule for the alert, select Add Rule Window from the Active Periods table menu.

The Add Active Period dialog appears.

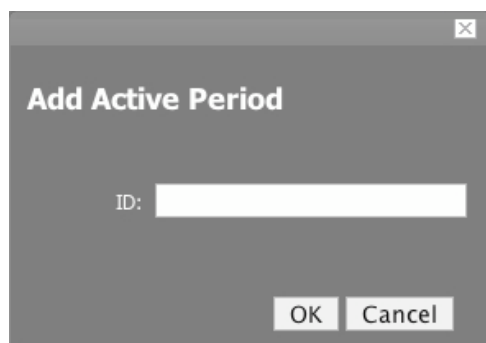


Figure 2.26. Add Active Period

4. Enter a name for the schedule in the ID field, and then click **OK**.

The Schedule you added appears in the Active Periods list.

5. Click the name of the new Schedule to set the details for the schedule.

The Schedule Details page appears.

Figure 2.27. Alerting Rules - Schedule Details

6. If you want to restrict this Alert to only monitor at certain times for certain durations, set the Enabled field to True.
7. In the Start area, enter the date you want the alert to start, or click the Select button to choose the date from a calendar.
8. In the fields to the right of the date, select an hour and minute for the Alert to start.
9. Use the Duration area to specify the length of time you want to Alert to be listening based on the start time.
10. If you want the Alerting period to repeat you can choose a time frame from the Repeat pop-up menu. You can choose from:
 - Never
 - Daily
 - Every Weekday
 - Weekly
 - Monthly
 - First Sunday of the Month
11. Choose a number of times to repeat the selected interval.
12. Click Save.

You have now saved all of the options for creating a new alert.

2.6.3. Escalation of Alerting in Zenoss

You can create an alerting hierarchy by using some of the different management tools available in Zenoss.

2.6.3.1. Creating an Alerting Hierarchy

You can create an alert hierarchy based on event status and delays by using alerting rules.

Sample Scenario:

You want to set up alerting rules so if that "Person A" (the first person in the hierarchy responding to alerts) does not acknowledge or suppress an event of a specific priority within a specific length of time (changing the event status), then "Person B" is notified by email to respond.

Step 1: Create an Alerting Rule for the Default Case (Initial State)

The default case is "when any new event of any priority occurs, alert Person A."

Create an alerting rule with a Delay value of 0 (zero) seconds.

Step 2: Create an Alerting Rule for the Next Level

For the next level in the hierarchy, the case is "If Person A does not acknowledge or suppress the event within an hour, then send an alert to the next person in the hierarchy (Person B)."

Create an additional alerting rule for Person B. To do this, you can:

- Create an additional rule for the currently logged-in User account.
- Add Person B's email address to the Address field. This address overrides the User account email.

Set the value of Delay to the number of seconds you want to wait after an event has come in to Zenoss but whose status has not changed. In this example, the wait time is one hour (3600 seconds).

In the Add filter area, select Event State, and then select the event state that will keep this rule from being executed on all events (including those acknowledged by Person A). For this example, select New.

2.6.4. Adding Delay and Schedules to Alerting Rules

You can use delays when creating alerting rules to set up on-call schedules and elevation hierarchies. Using delays will allow you to specify that if an event is not acknowledged in a certain amount of time, then Zenoss should send email to the next person in the hierarchy. You accomplish this by filtering on event state ('New') and adding a delay. Create an alerting rule for the tier 1 support person that does not have a delay so they find out immediately and can acknowledge the event if possible.

1. Create a second alerting rule (this one will be for the tier 2 person in the hierarchy) and enable it.
2. Use the 'where' clause to indicate that this rule is in effect only for events that have not yet been acknowledged.

Delay = 300 (in seconds, 5 minutes)

In the Where area,

Production State = Production

Severity >= Error

Event State = New

This rule now says fire this alert if there is an event in the system that is New (not acknowledged) for 5 minutes send email to this user.

3. Click the Message tab and in the Message (or subject) field enter the following:

[Zenoss-delayed] %(device)s %(summary)s

4. In the Clear message (or Subject) area, enter the following.

[Zenoss-delayed] CLEAR: %(device)s %(clearOrEventSummary)s

5. Click the Schedule tab to edit the schedule. You can tell the rule to only be active when this user is on call (remember each alerting rule is user based).
6. In the Add field enter a name for the new schedule.

The new schedule appears in the list.

7. Click the name of the new schedule and set these values:
 - **Name** - Name of the new schedule.
 - **Enabled** - Set to True.
 - **Start** - Specify when you want the rule to start.
 - **Duration** - Specify how long you want the rule to be in effect.
 - **Repeat** - Specify the number of times to repeat the schedule.
 - **Every** - Specify how many time periods to repeat.
8. Click **Save**.

2.7. Creating Custom Event Views

You can create and edit custom event views, narrowing the event list view according to filters you set and save. Custom event views are set individually for users.

To create a custom event view:

1. Click the Preferences link at the top right of the dashboard.
2. Click the Event Views tab.

The Event Views tab appears.

3. From the Event View table menu, select Add Event View.

The Add Event View dialog appears.

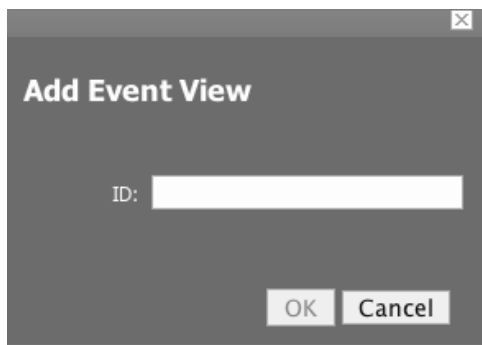


Figure 2.28. Add Event View Dialog

4. In the ID field, enter a name for the event view.
5. Click **OK**.

This custom event view appears in the list. Note that there is a custom alerting rainbow for this event view.

6. Click the link for the new event view you created.

Notice the size of the list and the number of entries.

7. Click the Edit tab.

The Edit Event views tab appears.

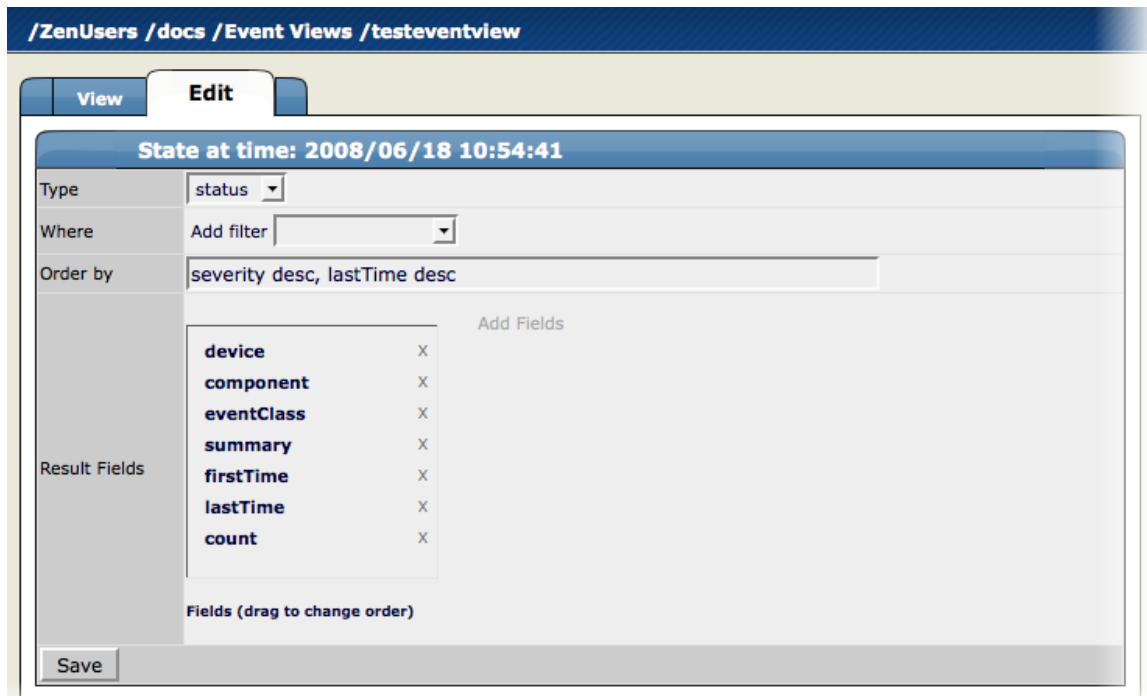


Figure 2.29. Custom Event View (Edit Tab)

8. Add conditions for this event view:

- **Type** - Select whether to show active events or the event history.
- **Where** - Use this area to add filters (similar to alerting rules "where" clauses).
- **Order by** - Specify the order of entries in the view.
- **Result Fields** - Select the fields to display in the view. Click the X next to each field you want to remove from the view.

9. Click **Save**.

You can click the View tab to see the results of the custom event view.

Chapter 3. Discovering and Modeling Devices

Modeling is the process by which Zenoss:

- Populates the device database
- Collects information about the devices in the system (such as operating system type or file system capacity)

Zenoss models devices when they are added to the Zenoss database, either manually or through the discovery process.

3.1. How Does Zenoss Model Devices?

To model devices, Zenoss can use:

- SNMP
- SSH
- WMI
- Telnet

The modeling method you select depends on your environment, and on the types of devices you want to model and monitor.

3.1.1. ZenModeler Daemon

Zenoss employs the zenmodeler daemon to model devices. The zenmodeler daemon iterates over the list of devices in the system and attempts to discover components -- such as network interfaces, file systems, process, and IP services -- of each device.

By default, Zenoss remodels each known device every 720 minutes (12 hours). You can change this interval by editing the value of Modeler Cycle Interval in the collector's configuration.

For larger deployments, modeling frequency may impact performance. In such environments, Zenoss recommends running the modeling process once daily from a cron job.

Read the following sections for information and procedures about:

- Adding devices to Zenoss
- Using Zenoss to discover devices on your network

3.2. Add a Single Device

Follow these steps to use the advanced add facility to add a single device to Zenoss.

1. From the Navigation menu, select Add Device.

The Add Device page appears.

Figure 3.1. Add Device

2. Enter information or make selections to add the device:

- **Device Name** - Enter the network (DNS) name or IP address of the device.
- **Device Class Path** - Select a device class to which this device will belong. For example, if the new device is a Windows server, then choose /Server/Windows/WMI.
- **Discovery Protocol** - Select a discovery protocol (auto or none).

i Device Name, Device Class Path, and Discovery Protocol are the only required fields to add the device. Zenoss recommends that you continue without adding more information or making selections, as information you enter or select may conflict with information Zenoss discovers about the device.

An exception to this is if you are adding a Cisco router in a device class other than /Network. In this case you should set the zProperty for zIfDescription to True. This will give you additional information about Cisco routers. By default, this option is set to True for the /Network class.

3. Scroll to the bottom of the page, and then click **Add Device**.

A status page appears, showing a log of the operations Zenoss is using to gather information about the device.

4. Scroll to the bottom of the status page, and then click the link that appears, similar to:

Navigate to device `DeviceName`

The Main Device page appears, showing the Status tab.

/Devices /Server /Linux /build.zenoss.loc

Status OS Hardware Software Events Perf Edit

Device Status

Device: **build.zenoss.loc** IP: **10.175.211.17** Status: **Up**

| Component Type | Status |
|-----------------|--------|
| Other | |
| /usr/sbin/snmpd | |
| IpRouteEntry | |
| IpInterface | |
| OSProcess | |
| FileSystem | |

Device Information

Organizers

Location: [/Annapolis/275 West Street/204](#)

Groups: [/Support/Blue Team](#)

Systems: [/Internet](#)

Collector: [localhost](#)

Ip Realm: None

OS

Tag #

Serial #

HW Make: [Generic](#)

HW Model: [Net-SNMP Agent](#)

OS Make: [RedHat](#)

OS Version: [Linux 2.6.9-34.0.2.ELsmp](#)

Rack Slot: 0

sysName: build.zenoss.loc

Contact: root@localhost

Location: Unknown

SNMP Descr: Linux build.zenoss.loc 2.6.9-34.0.2.ELsmp #1 SMP Fri Jul 7 19:52:49 CDT 2006 i686

Comments

Links

Figure 3.2. Main Device Page

3.2.1. Add a Device Within a Device Class

By default, when you add a device, Zenoss places it in the /Discovered class. You can choose instead to add a device directly to a specific device class in the hierarchy.

To add a device within a specific device class:

1. Navigate to the location in the devices hierarchy where you want to add the device.
2. Open the page menu, and then select Manage > Add Device.

The Add Device dialog appears.

Add a Device:

Name or IP:

Device Class: [/Network](#)

Discovery Protocol: [snmp](#)

Snmp Community:

Snmp Port: [161](#)

OK Cancel

Figure 3.3. Add Device Dialog

3. Complete the Name and Discovery Protocol fields. (For descriptions of valid values for these fields, refer to the section titled "Add a Single Device.") the Device Class value is the class you selected in the devices hierarchy.
4. Click **OK**.


The Device is added into the selected device class. The main Device page appears, showing the Status tab.

3.2.2. Add a Device - Alternate Method

You also can use the "easy add" facility to add a device to Zenoss. From the Add Device page, click "Easy Add" (located at the top right of the page). For information and instructions to help you add devices this way, refer to the section titled "Add Devices" in *Zenoss Getting Started*.

3.3. Discover Devices

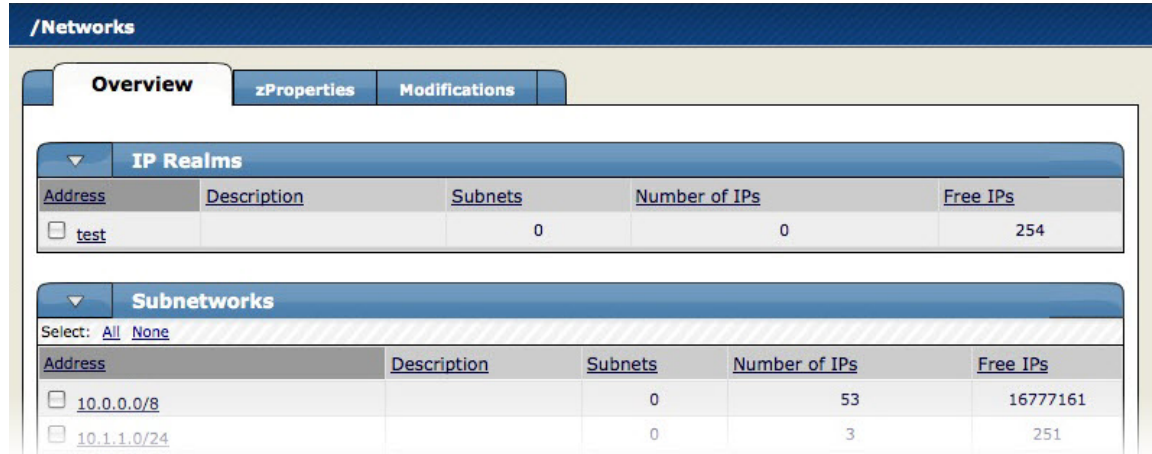
Zenoss' network discovery process iterates through every IP address in the sub-network ranges you specify, adding each device that responds to a ping request. Further, any device that responds to an SNMP request will have additional information added to it.

 To perform discovery, the machine on which Zenoss is installed must have an SNMP agent running.

To add all of the devices on a sub-network to Zenoss:


1. From the Navigation menu, select Networks.

The Networks page appears, displaying all of the currently configured sub-networks.



| /Networks | | | | |
|--|-------------|---------|---------------|----------|
| Overview zProperties Modifications | | | | |
| IP Realms | | | | |
| Address | Description | Subnets | Number of IPs | Free IPs |
| <input type="checkbox"/> test | | 0 | 0 | 254 |
| Subnetworks | | | | |
| Select: All None | | | | |
| Address | Description | Subnets | Number of IPs | Free IPs |
| <input type="checkbox"/> 10.0.0.0/8 | | 0 | 53 | 16777161 |
| <input type="checkbox"/> 10.1.1.0/24 | | 0 | 3 | 251 |

Figure 3.4. Networks Page (Overview Tab)

-  If the sub-network that you want to scan does not appear, then select Add Network from the Sub-networks table menu, and then supply the sub-network IP address and subnet mask (for example, 192.168.1.0/24).
2. Select one or more sub-networks that you want to scan for devices.
 3. Open the Subnetworks table menu, and then select Discover Devices.

The Discover Device page appears. This page shows the status of all the device collections in progress. (Do not navigate away from this page during discovery.)

| /Networks /192.168.17.0 | | | |
|-------------------------|---------|----------------|--|
| Discover Devices | | | |
| Time | Level | Module | Message |
| 2008-06-18 13:25:12 | INFO | zen.Utils | Executing command: /opt/zenoss/bin/zendisc run --weblog --net 192.168.17.0 |
| 2008-06-18 13:25:15 | WARNING | zen.ZenDisc | Reconnected to ZenHub |
| 2008-06-18 13:25:15 | INFO | zen.ZenDisc | connected to ZenHub |
| 2008-06-18 13:25:15 | INFO | zen.ZenDisc | discover network '192.168.17.0' |
| 2008-06-18 13:26:05 | INFO | zen.ZenDisc | discovered 4 active ips |
| 2008-06-18 13:27:11 | INFO | zen.ZenDisc | Result: Discovered 4 devices |
| 2008-06-18 13:27:12 | INFO | zen.ZenDisc | no wmi plugins found for 192.168.17.2 |
| 2008-06-18 13:27:12 | INFO | zen.ZenDisc | no python plugins found for 192.168.17.2 |
| 2008-06-18 13:27:12 | INFO | zen.ZenDisc | no cmd plugins found for 192.168.17.2 |
| 2008-06-18 13:27:12 | INFO | zen.ZenDisc | snmp collection device 192.168.17.2 |
| 2008-06-18 13:27:12 | INFO | zen.ZenDisc | plugins: zenoss.snmp.NewDeviceMap, zenoss.snmp.DeviceMap, zenoss.snmp.InterfaceMap, zenoss.snmp.RouteMap |
| 2008-06-18 13:27:12 | INFO | zen.ZenDisc | no portscan plugins found for 192.168.17.2 |
| 2008-06-18 13:27:20 | INFO | zen.SnmpClient | Device timed out: SNMP info for 192.168.17.2 at 192.168.17.2:161 timeout: 2.5 tries: 2 version: v1 community: public |
| 2008-06-18 13:27:20 | INFO | zen.SnmpClient | snmp client finished collection for 192.168.17.2 |
| 2008-06-18 13:27:20 | WARNING | zen.SnmpClient | Device 192.168.17.2 timed out: are your SNMP settings correct? |
| 2008-06-18 13:27:20 | INFO | zen.ZenDisc | no change detected |
| 2008-06-18 13:27:20 | INFO | zen.ZenDisc | no wmi plugins found for 192.168.17.3 |
| 2008-06-18 13:27:20 | INFO | zen.ZenDisc | no python plugins found for 192.168.17.3 |
| 2008-06-18 13:27:20 | INFO | zen.ZenDisc | no cmd plugins found for 192.168.17.3 |

Figure 3.5. Discover Device

Zenoss first models the monitoring machine, and then walks through the routing tables of all routers it locates. Discovery continues while valid SNMP access is found or until a network is discovered in the DMD that has its `zAutoDiscover` property set to `False`.

Zenoss places routers discovered through this process in the device path `/Network/Router`. Devices are placed in the `/Discovered` device class.

3.3.1. Discover Devices - Alternate Method

You also can use the "easy add" facility to discover devices. From the Add Device page, click "Easy Add" (located at the top right of the page). For information and instructions to help you discover devices this way, refer to the section titled "Add Devices" in *Zenoss Getting Started*.

3.3.2. Classifying Discovered Devices

Once discovery is complete, you must move discovered devices (placed, by default, in the `/Discovered` class) to an appropriate device class in the hierarchy. Moving devices to their correct hierarchy location initiates the monitoring process.

In general, servers are organized by operating system. If Zenoss discovers Windows devices, for example, you might choose to relocate them to `/Server/Windows/WMI`. Similarly, you might choose to classify discovered Linux devices in the `/Server/Linux` device class.

To classify discovered devices:

1. Select one or more discovered devices in the device list.

2. From the page menu, select Set to Class.
3. In the Edit Device Class dialog, select a device class.
4. Click **Move**.

3.3.3. Authenticating Devices

For each device that is added to the Zenoss database and set to its proper device class, Zenoss may require authentication information before it can gather device information and monitor the device.

For example, for a device in the /Server/Windows/WMI class, you must supply your Windows user name and password before Zenoss can monitor the device. To do this:

1. Navigate to the device in the device list.
2. From the page menu, select More > zProperties.
3. Set the user name and password values in the zWinUser and zWinPassword zProperties.
4. Click **Save**.

Similarly, for a device in the /Server/SSH/GenericLinux class, you must supply your SSH user name and password. Set these values in the device's zCommandUsername and zCommandPassword zProperties.

3.3.4. Adding Information to a Device Record

You may want to add details about a discovered device, such as its Business System or Location. To add information, select the device in the list, and then go to the Edit tab.

3.4. Modeling Devices Using SNMP

Read this section for information about the methods Zenoss uses to model devices using SNMP.

3.4.1. Testing to See if a Device is Running SNMP

To test whether a device is running SNMP, run this command:

```
$ snmpwalk -v1 -c communityString DeviceName system
```

If this command does not time out, then SNMP is installed and working correctly.

3.4.2. Modeling Remote Windows Devices Using SNMP

By default, Windows may not have SNMP installed. To install SNMP, follow these general steps:

1. Go to Start > Control Panel > Add or Remove Programs > Add/Remove Windows Components.
2. Select the option for Management and Monitoring tools and install them.
3. Go to Control Panel > Administrative Tools > Services and start the SNMP Service and SNMP Trap Service.
4. Set the SNMP Community string in the SNMP Service properties to the community string of your SNMP.

If you want processor and memory monitoring, install SNMP-Informant on the device. Go to <http://www.snmp-informant.com> and download SNMP for Windows.

To collect Windows Event logs or log files from a Windows box using syslog, you can use the SyslogAgent Windows add-on, available from:

<http://syslogserver.com/syslogagent.html>

3.4.3. Modeling Remote Linux Devices Using SNMP

To configure a Linux machine for monitoring, it must have SNMP installed. A good Linux SNMP application is net-snmp. Download, install, and configure net-snmp to then use SNMP to monitor Linux devices with Zenoss.

3.4.4. Modeling Cisco Devices Using SNMP

Cisco devices come with SNMP already installed. However, you have to configure SNMP on each Cisco device to be in the same community as the rest of your network.


3.5. Modeling Using SSH/COMMAND

Modeling also can be done by running commands on the remote device and interpreting the results. This provides an easier, more scalable, and more flexible way to gather information that may not be available through any other means.

Each built-in Zenoss modeling command plugin is differentiated by the platform on which it runs. To determine the platform for the device you want to model, run the **uname** command in a shell on the device.

To model a device using command plugins, first add the device to Zenoss by using the protocol "none," and then choose the plugins you want to apply:

1. Go to the Add Device page.
2. Set discover to a value of None.
3. After adding the device, navigate to the device and view its zProperties tab.
4. If necessary, set zCommandUsername and zCommandPassword to the user name and password of the device (or set up authentication by using RSA/DSA keys.)

 If using RSA keys for a device or device class, change the value of the zKeyPath zProperty to:

```
~/.ssh/id_rsa
```

5. Click Edit next to zCollectorPlugins.
6. Click Add Fields for a complete list of command plugins.
7. Make sure zenoss.cmd.uname is positioned first on the left side.
8. Click the X next to plugins you wish to remove, and then drag remaining plugins to the left.
9. Remodel the device.

3.5.1. Using Device Class to Monitor Devices Using SSH

The /Server/Cmd device class is an example configuration for modeling and monitoring devices using SSH. The zCollectorPlugins have been modified (see the section titled "Modeling Using SSH/Command"), and the device, file system, and Ethernet interface templates used to gather data over SSH have been created. You can use this device class as a reference for your own configuration; or, if you have a device that needs to be modeled or monitored via SSH/Command, you can place it in this device class to use the pre-configured templates and zProperties. You also must set the zCommandUsername and zCommandPassword zProperties to the appropriate SSH login information for each device.

3.6. Modeling Devices Using Port Scan

You can model IP services by doing a port scan, using the Nmap Security Scanner (<http://nmap.org/>). You must provide the full path to your system's `nmap` command.

To determine where `nmap` is installed, at the command line, enter:

```
which nmap
```

If your system returns a result similar to:

```
/usr/bin/which: no nmap in  
(/opt/zenoss/bin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/opt/zenoss/bin)
```

then `nmap` is not installed. Install it, and then try again.

After locating the `nmap` command (including the directory beginning with `/`), enter the following as the `zenoss` user on the Zenoss server:

```
cd $ZENHOME/libexec
ln -s Full_Path_to_nmap
```

To model a device using a port scan:

1. Go to the `zProperties` tab of a device.
2. Change the `zTransportPreference` to "portscan."
3. Remodel the device.

3.6.1. Using the /Server/Scan Device Class to Monitor with Port Scan

The `/Server/Scan` device class is an example configuration for modeling devices by using a port scan. You can use this device class as a reference for your own configuration; or, if you have a device that will use only a port scan, you can place it under this device class and remodel the device.

3.7. Collector Plugins

Zenoss uses plug-in maps to map real world information into the standard Zenoss model. Input to the plug-ins can come from SNMP, SSH or Telnet. Selection of plug-ins to run against a device is done by matching the plug-in name against the `zCollectorPlugins` `zProperty`. Plug-ins selected in `zCollectorPlugins` are the ones that are collected.

- **DeviceMap** – Collects basic information about a device, such as its OS type and hardware model.
- **InterfaceMap** – Collects the list of interfaces on a device.
- **RouteMap** – Collects the routing table from the device.
- **IpServicesMap** – Collects the IP services running on the device.
- **FileSystemMap** – Collects the list of file systems on a device.

3.7.1. Viewing Collector Plugins for a Device

Plugins are controlled by a regular expressions that match their names. To see a list of plugins for any device:

1. Navigate to the device.
2. From the page menu, select `More > Collector Plugins`.

The `Collector Plugins` tab appears, showing all of the collector plugins for the device.

3.8. Debugging the Modeling Process

You can run the Zenoss modeler from the command line against a single device. This feature is useful when debugging issues with a plugin.

By passing the `--collect` command to the modeler, you can control which modeler plugins Zenoss uses. For example, the following command runs only the interface plugin against the `build.zenoss.loc` device:

```
$ zenmodeler run -v 10 --collect=IpInterface -d build.zenoss.loc
```

If the command returns any stack traces, Zenoss recommends that you forward these details to Support for assistance:

- Command you ran
- Stack trace or stack traces returned
- Version of your Zenoss instance
- OS version and patch level for the remote device

Chapter 4. Working with Devices

This chapter provides information and procedures for managing devices in Zenoss.

4.1. Device List

The Device List shows all devices in the system. From this view, you can search for devices and perform a range of device management tasks.

To access the device list, select Device List from the navigation menu.

| Device Id | IP | Class | Prod State | Event |
|---|----------------|-------------------------|----------------|----------|
| <input type="checkbox"/> annapolis.md.bad.comcast.net | 68.87.136.205 | /Network/Router/Phantom | Production | 0/0 0/0 |
| <input type="checkbox"/> apc1.zenoss.loc | 192.168.1.51 | /Power/UPS/APC | Decommissioned | 0/0 0/0 |
| <input type="checkbox"/> appcorebeta.zenoss.loc | 10.175.211.215 | /Server/Linux | Production | 1/1 0/0 |
| <input type="checkbox"/> appentbeta.zenoss.loc | 10.175.211.216 | /Server/Linux | Production | 1/1 0/0 |
| <input type="checkbox"/> build.zenoss.loc | 10.175.211.17 | /Server/Linux | Production | 0/0 0/1 |
| <input type="checkbox"/> buildmaster.zenoss.loc | 10.175.211.90 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> cent4b-64.zenoss.loc | 10.175.211.211 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> cent4b.zenoss.loc | 10.175.211.210 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> cent4t-64.zenoss.loc | 10.175.211.68 | /Server/Linux | Production | 0/0 0/3 |
| <input type="checkbox"/> cent4t.zenoss.loc | 10.175.211.66 | /Server/Linux | Production | 0/0 0/2 |
| <input type="checkbox"/> cent5-java.zenoss.loc | 10.175.211.93 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> cent5b-64.zenoss.loc | 10.175.211.213 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> cent5b.zenoss.loc | 10.175.211.212 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> cent5c.zenoss.loc | 10.175.211.94 | /Server/Linux | Production | 0/0 0/12 |
| <input type="checkbox"/> cent5t-64.zenoss.loc | 10.175.211.69 | /Server/Linux | Production | 0/0 0/1 |
| <input type="checkbox"/> cent5t.zenoss.loc | 10.175.211.73 | /Server/Linux | Production | 0/0 0/2 |
| <input type="checkbox"/> cqibbons-dev.zenoss.loc | 10.175.211.233 | /Server/Linux | Production | 0/0 0/0 |
| <input type="checkbox"/> collect2.zenoss.loc | 10.175.211.236 | /Server/Linux | Production | 0/0 0/0 |

Figure 4.1. Device List

To view a single device, click its name in the list. The Device page appears.

4.1.1. Device Page Tabs

Zenoss assigns properties and attributes to each device in the system. It categorizes this information and makes it available from tabs located at the top of the Device page.

4.1.1.1. Status Tab

The Status tab appears when you select a device from the device list.

The screenshot shows the Zenoss Device Page for the device **build.zenoss.loc** with IP **10.175.211.17**. The status is **Up**. The left sidebar contains navigation links for Main Views, Classes, Browse By, and Management. The top navigation bar includes tabs for Status, OS, Hardware, Software, Events, Perf, and Edit.

Device Status

Device: **build.zenoss.loc** IP: **10.175.211.17** Status: **Up**

| Component Type | Status |
|-----------------|--------|
| Other | |
| /usr/sbin/snmpd | |
| IpRouteEntry | |
| IpInterface | |
| OSProcess | |
| FileSystem | |

Device Information

| Organizers | OS |
|--|--|
| Location: /Annapolis/275 West Street/204 | Tag # |
| Groups: /Support/Blue Team | Serial # |
| Systems: /Internet | HW Make: Generic |
| Collector: localhost | HW Model: Net-SNMP Agent |
| Ip Realm: None | OS Make: RedHat |
| | OS Version: Linux 2.6.9-34.0.2.ELsmp |
| | Rack Slot: 0 |
| | sysName: build.zenoss.loc |
| | Contact: root@localhost |
| | Location: Unknown |

SNMP Descr: Linux build.zenoss.loc 2.6.9-34.0.2.ELsmp #1 SMP Fri Jul 7 19:52:49 CDT 2006 i686

Comments

Links

Figure 4.2. Device Page (Status Tab)

Device Status Table

The Device Status table provides important device status at a glance. Events, grouped by severity, are found at the left side of this table. Click the indicators in this "event rainbow" to view the list of events for the device.

Key information appears at the top of the Device Status table:

- **Device** - Displays the device name.
- **IP** - Displays the device IP address. Zenoss uses this address to communicate with the device.
- **Status** - Shows the current results of the ping test.

Additional device status information appears to the right of the event rainbow:

- **Availability** - Shows seven-day availability, as defined by the availability of the device by measuring ping.

Zenoss determines availability by taking all events of type `/Status/Ping`, with a Severity 5 and higher for the past seven days, and calculating the amount of time these systems have been in a down state, as follows:

$$(((1\ 248\ 400\ 328 - 1\ 247\ 795\ 528) / 60) / 60) / 24 = 7$$

Duration is specified by the Default Availability Report (accessible from the Event Manager area).

- **Uptime** - Duration the device has been "up" and running, as reported by the agent on the device. This information can be acquired through SNMP, WMI, or SSH.
- **State** - Set this value from the Device page Edit tab. Indicates one of these device states:
 - **Production** - Zenoss monitors devices in production, reporting issues on the dashboard and sending event notifications.
 - **Maintenance** - Zenoss collects data for devices in maintenance, but does not report issues or send event notifications.

- **Decommissioned** - Zenoss does not monitor the device.
- **Priority** - Ranks the importance of your devices. You can use priority settings to control alerts. Set this value from the Device page Edit tab.
- **Locks** - Prevent the modeler from overriding custom changes. Set locks from the Device page menu, from the Manage > Lock Devices selection.

For more information about locking devices, refer to the section titled "Locking Device Configuration."

- **Last Change** - Displays the latest time the modeler detected a change on the device and updated its entry in Zenoss.
- **Last Collection** - Displays the latest time the modeler checked for changes to the device.
- **First Seen** - Displays the date and time the device was added to Zenoss.

Component Status

On the right side of the Device Status table is the Component Status list. Each item in the list is a type of device component, such as IPService, WinService, IpRouteEntry, IpInterface, CPU, and FileSystem. Click the indicators in the Status column to view the event page for that component.

The status of each device component type, as shown by the color of its indicator, is determined by the collective status of the monitored components of the same type. For example, if the IpService status is green, then all monitored IpServices on this device are functioning normally. If there is an event related to a monitored IpService, then the component and highest severity event associated with that component are displayed in the status.

If there is an event unrelated to a known component, then Zenoss places it in the component type Other.

Device Information

The Device Information area provides system information and details about the device's organizers.

- **Organizers** - Identifies the groups in which you have included this device. It also displays values for:
 - **Collector** - If your Zenoss implementation uses a single collector, then this value is "localhost." If your Zenoss implementation uses distributed collectors, then this value is the name of the collector that monitors the device.
 - **IP Realm** - Specifies an advanced configuration value for the MultiRealmIP ZenPack.
- **OS** - Shows system information returned by the device collectors. The details provided in this area vary, depending on data collection method.

The lowest section of the Device Information area includes Links, which displays links between this device and other external systems. Links are implemented by the zlinks zProperty.

For more information about custom links, see the chapter titled "Properties and Templates."

4.1.1.2. OS (Operating Systems) Tab

The OS tab shows a detailed breakdown of all of the operating system components found by the modeling process. For each of these components, you can see current status, names, and lock status.

The screenshot displays the Zenoss Device Page for a Virtual Machine Host (ESX) at the path `/Devices /Server /Virtual Machine Host /ESX /esx1.zenoss.loc`. The interface is divided into a left sidebar with navigation menus and a main content area with several tabs.

Main Views: Dashboard, Event Console, Device List, Network Map.

Classes: Events, **Devices** (selected), Services, Processes, Products.

Browse By: Systems, Groups, Locations, Networks, Reports.

Management: Add Device, Mibs, Collectors, Settings, Event Manager.

OS Tab: The main content area has tabs for Status, OS (selected), Guests, Hardware, Software, Events, Perf, and Edit.

Interfaces: A table listing network interfaces with columns for Name, IP Address, Network, MAC, O (Operational), A (Administrative), and Lock. The table shows several interfaces, including `lo`, `vmnic0`, `vmnic1`, `vmnic2`, `vmnic3`, and `vswif0`.

Win Services: A table listing Windows services with columns for Caption, StartMode, StartName, Name, Status, and Lock. The table is currently empty.

OS Processes: A table listing OS processes with columns for Class, Name, Restarts, Fail Severity, Status, and Lock. The table is currently empty.

IP Services: A table listing IP services with columns for Name, Proto, Port, Ips, Description, Status, and Lock. The table shows one service, `http`, which is monitored.

File Systems: A table listing file systems with columns for Mount, Total bytes, Used bytes, Free bytes, % Util, and Lock. The table shows three file systems: `/`, `/boot`, and `/var/log`.

Routes: A table listing routes with columns for Destination, NextHop, Interface, Protocol, Type, and Lock. The table shows three routes: `0.0.0.0/0`, `10.175.211.0/24`, and `169.254.0.0/16`.

Figure 4.3. Device Page (OS Tab)

Monitored Selection

Some areas of the OS tab feature a Monitored option, which filters the list of interfaces shown in that area. To view all interfaces, de-select this option. If you want to view only those interfaces that Zenoss is monitoring, then select the option.

Interfaces

The Interfaces area shows basic information about each of the logical and physical network interfaces that Zenoss models.

In the IP Address column, addresses may be links to additional information in the network information database.

The red and green status indicators in this area provide at-a-glance information about the status of each network listed:

- **O** - Indicates that the network is operationally online, meaning that the network interface is operating when the indicator shows green.
- **A** - Indicates whether the network is administratively up, meaning that it has been configured to operate when the indicator shows green.
- **M** - If the indicator is green, Zenoss is monitoring this network; if red, then Zenoss is not monitoring it.

Win Services

This area shows detailed information about the status of Windows-based services. Key information in this area includes:

- **StartMode** - Windows services with a start mode of disable or manual are monitored by Zenoss.
- **StartName** - Indicates the user under which the service runs.

OS Processes

Shows OS processes for this device. After adding an OS process, you should re-model the device.

The Restarts column shows whether Zenoss will generate an event if the process is detected as restarted.

IP Services

The IP Services area shows the TCP and UDP ports that are currently listening on a device. Zenoss monitors only TCP ports.

Make sure you have TCP connectivity between your Zenoss server and the monitored server.

Key information in this area includes:

- **Ips** - In most cases, shows 0.0.0.0, which indicates that the service is listening on all IP addresses.
- **Status** - If the status indicator is green, then the TCP port check succeeded; if red, then it did not succeed.

File Systems

This area lets you view file system status, if file system monitoring is enabled. To enable file system monitoring, select Monitoring from the File Systems table menu, and then select the Enable option in the dialog.

File systems can be monitored via SNMP only if the system has a valid HOST-RESOURCES MIB.

If "Unknown" appears in the Used bytes, Free bytes, or % Util columns, then performance collection may not yet have begun.

Zenoss monitors the amount of blocks used and shows:

- Total bytes
- Available bytes
- Used bytes
- Percentage used

Routes

Routes are collected to model the network topology so that root cause analysis can be determined by the Zenoss server. Routes are not monitored.

4.1.1.3. Hardware Tab

The Hardware tab shows a detailed breakdown of all of the hardware components found by the modeling process. This area might provide information about the device's available and used memory, available and used swap space, CPUs, hard disks, expansion cards, fans, temperature sensors and power supplies.

The information show in this area varies depending on the device type.

The screenshot displays the Zenoss Device Page for a server named `/win2003.zenoss.loc`. The **Hardware** tab is selected, showing various system components:

- Memory:** 511.3MB, Swap: 1.2GB
- CPUs:** A table with columns: Socket, Manufacturer, Model, Speed, Ext Speed, L1, L2, Volts. One entry is shown: PROC, Unknown, GenuineIntel x86 Family 15 Model 4 Stepping 9, 2533 MHz, 533 MHz, 16 KB, 256 KB, 1500 mV.
- Hard Disks:** A table with columns: Name, Snmp Index. One entry is shown: C, 2.67.58.
- Expansion Cards:** A table with columns: Slot, Manufacturer, Model. Nine entries are listed, including Intel Corporation PCI Express Root Port, Intel Corporation I/O Controller Hub ICH7R UHCI USB, Intel Corporation I/O Controller Hub ICH7R PCI Express Port 1, Intel Corporation 82801GB GR ICH7 Family LPC Interface Bridge, Broadcom Corp BCM5721 NetXtreme Gigabit Ethernet PCI Express, and XGI Xabre Graphics Inc XGI GX20 Video Controller.

At the bottom, there is a pagination bar showing "1 of 7" items, a filter dropdown set to "Boxes modeled a..g WMI", and a "Page Size" of 40.

Figure 4.4. Device Page (Hardware Tab)

4.1.1.4. Software Tab

The Software tab lists software installed on the device. The details provided in this area depend on the method used to model the device.

Listed software links into Zenoss' inventory of software in your IT infrastructure. You can view this inventory from the Products link on the Navigation menu.

▲

/Devices /Server /Windows /WMI /win2003.zenoss.loc

Main Views

Dashboard

Event Console

Device List

Network Map

Classes

Events

Devices

Services

Processes

Products

Browse By

Systems

Groups

Locations

Networks

Reports

Management

Add Device

Mibs

Collectors

Settings

Event Manager

Status

OS

Hardware

Software

Events

Perf

Edit

Installed Software

| Manufacturer | Name | Install Date |
|----------------|--|---------------------|
| APC | APC PowerChute Business Edition Server | 2006/11/29 17:15:30 |
| Unknown | Active_UNDELETE 7 | 2008/05/15 12:52:20 |
| Adobe | Adobe Flash Player 9 ActiveX | 2006/09/13 12:16:06 |
| Unknown | Adobe Flash Player Plugin | 2007/12/11 15:01:52 |
| Unknown | Check Point VPN-1 SecureClient NG AI_R56 | 2008/02/05 08:25:18 |
| Unknown | Dell OpenManage Server Administrator | 2006/02/03 13:06:30 |
| HP | HP Software Update | 2006/10/26 10:40:24 |
| Microsoft | Internet Explorer Developer Toolbar | 2006/03/31 14:57:46 |
| Unknown | Java(TM) 6 Update 3 | 2007/12/11 14:58:48 |
| Unknown | Lexmark Software Uninstall | 2007/09/06 16:37:14 |
| Unknown | Lexmark X500 Series Network TWAIN Scan | 2007/09/06 16:37:16 |
| Unknown | Lexmark Z600 Series | 2006/10/25 09:45:32 |
| Unknown | MSXML 4.0 SP2 (KB936181) | 2008/01/17 09:30:46 |
| Unknown | Microsoft .NET Framework 2.0 Service Pack 1 | 2008/05/21 17:59:58 |
| Unknown | Microsoft Internationalized Domain Names Mitigation APIs | 2007/05/30 14:33:36 |
| Unknown | Microsoft National Language Support Downlevel APIs | 2007/05/30 14:33:36 |
| Unknown | Microsoft Office SharePoint Portal Server 2003 | 2007/12/28 10:20:46 |
| Microsoft | Microsoft SQL Server 2000 | 2007/04/09 16:37:06 |
| Unknown | Microsoft SQL Server Desktop Engine (SHAREPOINTPORTAL) | 2007/12/27 15:52:48 |
| Unknown | Microsoft Silverlight | 2008/05/21 17:52:06 |
| Microsoft | Microsoft Windows SharePoint Services 2.0 | 2007/12/28 10:17:20 |
| Unknown | Mozilla Firefox (2.0.0.14) | 2008/04/25 18:14:40 |
| Unknown | Python 2.4 pywin32-210 | 2007/06/14 11:36:06 |
| Python | Python 2.4.2 | 2006/02/16 15:48:36 |
| Unknown | SNMP Informant Agent_Advanced Edition | 2007/05/29 17:02:00 |
| Unknown | SNMP Informant Agent_SQLServer Edition | 2007/05/29 17:02:40 |
| Snmp-Informant | SNMP Informant Agent_Standard Edition | 2006/04/20 14:52:28 |
| Unknown | Security Update for CAPICOM_KB931906 | 2007/05/21 13:16:50 |
| Unknown | Security Update for CAPICOM_KB931906 | 2007/05/21 13:16:50 |
| Unknown | Security Update for Windows Internet Explorer 7_KB933566 | 2007/06/14 16:09:26 |
| Unknown | Security Update for Windows Internet Explorer 7 (KB937143) | 2007/08/16 10:30:32 |
| Unknown | Security Update for Windows Internet Explorer 7 (KB938127) | 2007/12/24 12:17:08 |
| Unknown | Security Update for Windows Internet Explorer 7 (KB939653) | 2007/10/12 20:28:04 |
| Unknown | Security Update for Windows Internet Explorer 7 (KB942615) | 2007/12/21 13:48:38 |
| Unknown | Security Update for Windows Internet Explorer 7 (KB944533) | 2008/02/28 13:56:04 |

Figure 4.5. Device Page (Software Tab)

4.1.1.5. Events Tab

The Events tab provides events information that is scoped to the device. From here, you can:

- Sort event information by a range of categories
- Classify and acknowledge events
- Filter events by severity, state, or by one of several categories

| /Devices /Server /Linux /zenoss.zenoss.loc / Event Console | | |
|--|----------|--|
| | | |
| Status | OS | Hardware |
| Software | | |
| Events | | |
| Perf | | |
| Edit | | |
| | | |
| Select Export Configure | | |
| Status | Severity | Component |
| Event Class | Summary | First Seen |
| ... | ... | ... |
| ! | ! | /Status/RPC |
| ! | ! | CRITICAL: RPC program nfs udp is not running |
| 2009-09-18 10:... | | |
| ✓ | ! | zenwin |
| ! | ! | /Status/Heartbeat |
| ! | ! | localhost zenwin heartbeat failure |
| 2009-09-09 10:... | | |
| ✓ | ! | zenstatus |
| ! | ! | /Status/Heartbeat |
| ! | ! | localhost zenstatus heartbeat failure |
| 2009-09-09 10:... | | |
| ✓ | ! | zenmodeler |
| ! | ! | /Status/Heartbeat |
| ! | ! | localhost zenmodeler heartbeat failure |
| 2009-09-09 10:... | | |
| ! | ! | zeneventlog |
| ! | ! | /Status/Heartbeat |
| ! | ! | localhost zeneventlog heartbeat failure |
| 2009-09-09 10:... | | |
| ! | ! | /Status/Jabber |
| ! | ! | Connection refused |
| 2009-09-18 10:... | | |
| ! | ! | /Perf/CPU |
| ! | ! | threshold of CPU Utilization not met: current value 0.02 |
| 2009-09-18 10:... | | |
| ! | ! | /opt/zenoss/bin/p |
| ! | ! | /Status/OSProces |
| ! | ! | Process not running: /opt/zenoss/bin/python /opt/zenoss/Products/ZenWin/zeneventlog.py --configfile /opt/zenos |
| 2009-09-18 10:... | | |
| ! | ! | zenwinperf |
| ! | ! | /Status/OSProces |
| ! | ! | Process not running: zenwinperf |
| 2009-09-18 10:... | | |
| ! | ! | /usr/sbin/httpd |
| ! | ! | /Status/OSProces |
| ! | ! | Process not running: /usr/sbin/httpd |
| 2009-09-18 10:... | | |
| ! | ! | /opt/zenoss/bin/p |
| ! | ! | /Status/OSProces |
| ! | ! | Process not running: /opt/zenoss/bin/python /opt/zenoss/Products/ZenHub/zenhubworker.py --configfile /opt/zeni |
| 2009-09-18 10:... | | |
| ! | ! | /opt/zenoss/bin/p |
| ! | ! | /Status/OSProces |
| ! | ! | Process not running: /opt/zenoss/bin/python /opt/zenoss/Products/ZenHub/zenhubworker.py --configfile /opt/zeni |
| 2009-09-18 10:... | | |
| ! | ! | /Perf/CPU |
| ! | ! | threshold of CPU Utilization not met: current value 0.02 |
| 2009-09-18 10:... | | |
| ! | ! | apache |
| ! | ! | /App/Apache |
| ! | ! | (111, 'Connection refused') |
| 2009-09-18 10:... | | |

Figure 4.6. Device Page (Events Tab)

For detailed information about the event console and how Zenoss handles events, see the chapter titled "Event Management."

4.1.1.6. Performance (Perf) Tab

The Perf tab shows performance graphs defined for the currently selected device.

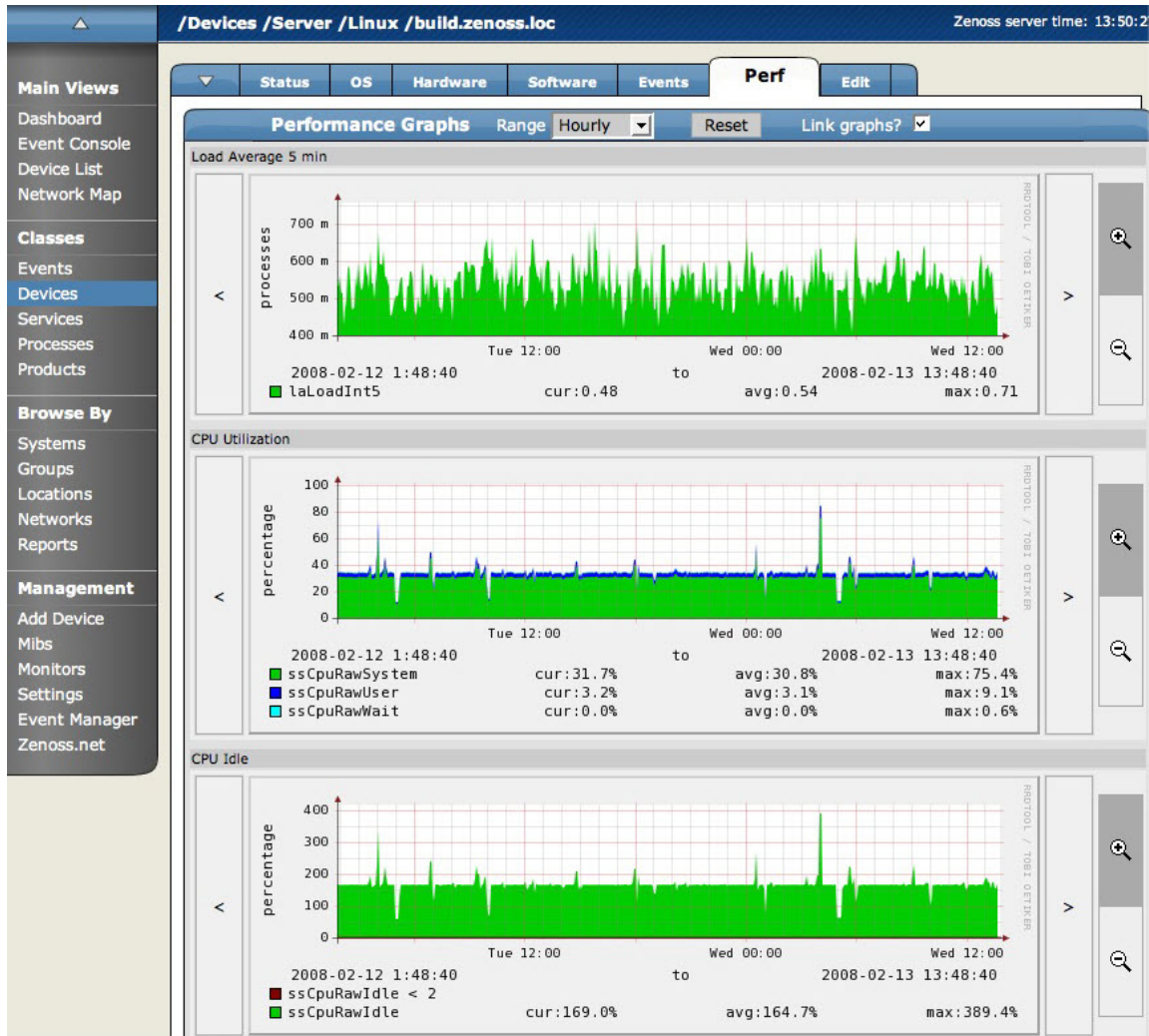


Figure 4.7. Device Page (Performance Tab)

You can use the arrow key and magnifying glass controls on the sides of each graph to change the graph view, scrolling through or zooming in or out of a graph.

From this tab, you can control these performance graph options:

- **Range** - Select the span of time displayed in the graph. You can select Hourly (past 36 hours), Daily (past 10 days), Weekly (past six weeks), Monthly (past 15 months), or Yearly (past two years).
- **Reset** - Click to return to the default (initial view) of the graphs.
- **Link graphs** - By default, all graphs move together. If you click the back arrow for a graph, for example, then all graphs move backward. De-select the Link graphs option to control each graph individually.
- **Stop** - Turns off automatic refresh of the graphs.

For more information about Zenoss performance monitoring and defining performance graphs, see the section titled "Performance Monitoring."

4.1.1.7. Edit Tab

Use the Edit tab to change properties on a device.

Edit Device Device State at time: 2008/06/18 11:11:52

Collector: localhost

Attributes

Snmp Community: Snmp Port: 161

Tag Number: Serial Number:

Production State: Production Priority: Normal

Rack Slot: 0

Comments:

Relations

HW Manufacturer: Generic Add

HW Product: Add

OS Manufacturer: RedHat Add

OS Product: Add

Location Path: /Annapolis/275 West Street/204

New Location: Add

Systems

- /
- /Buildbot
- /CRM
- /Customers
- /Customers/AT and T
- /Customers/AT and T/West
- /Customers/Disney
- /Development
- /Development/Build
- /Development/TestInstalls
- /Development/TestTargets
- /Email
- /Internet
- /Network
- /Services
- /testab
- /Testing
- /VmWareEsxServers
- /VmWareEsxServers/VmServer

New System: Add

Groups

- /
- /Admin 1 Group
- /build
- /Customers
- /Customers/a
- /Customers/D and E
- /Customers/D and E/West
- /Customers/D and E/West/Blah
- /Network
- /Support
- /Support/Blue Team
- /Support/Green Team

New DeviceGroup: Add

Save

Figure 4.8. Device Page (Edit Tab)

From this tab, you can change values for various collector attributes and relations.

4.2. Managing Devices and Device Attributes

Read the information and procedures in this section to learn how to manage devices in Zenoss.

4.2.1. Managing Custom Device Properties

You can use the Custom Properties page to create an unlimited number of custom commands. Custom commands are defined at the root level of the device tree and apply to all devices.

To access the Custom Properties page, open the Device page menu, and then select More > Custom.

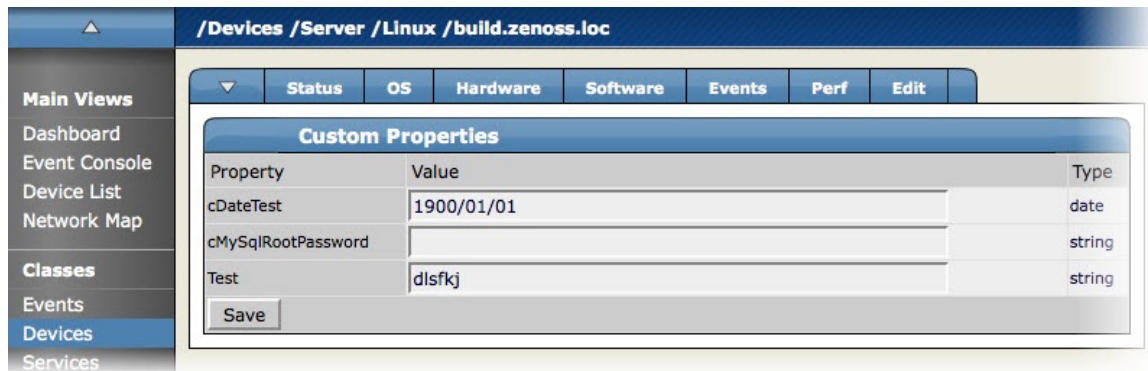


Figure 4.9. Device Page (Custom Properties)

4.2.2. Managing Device zProperties

From the Device zProperties page, you can configure zProperties for devices.

 For detailed information about working with zProperties, see the chapter titled "Properties and Templates."

To configure zProperties for multiple devices, click the zProperties tab from the Device Overview tab. To configure zProperties for an individual device, click the device name in the Device Overview, and then click the zProperties tab for that device.

To access the zProperties, open the device table menu, and then select More > zProperties.

| Property | Value | Type | Path |
|-------------------------|------------------------------|---------|--|
| zAxiPassword | ***** | string | / |
| zAxiUsername | administrator | string | / |
| zCollectorClientTimeout | 180 | int | / |
| zCollectorDecoding | latin-1 | string | / |
| zCollectorLogChanges | True | boolean | / |
| zCollectorPlugins | Edit | lines | /Server/Linux |
| zCommandCommandTimeout | 15.0 | float | / |
| zCommandCycleTime | 60 | int | / |
| zCommandExistenceTest | test -f %s | string | / |
| zCommandLoginTimeout | 10.0 | float | / |
| zCommandLoginTries | 1 | int | / |
| zCommandPassword | | string | / |
| zCommandPath | /opt/zenoss/libexec | string | / |
| zCommandPort | 22 | int | / |
| zCommandProtocol | ssh | string | / |
| zCommandSearchPath | | lines | / |
| zCommandUsername | | string | / |
| zDeviceTemplates | Device DnsMonitor test | lines | /Server/Linux/devices /build.zenoss.loc |

Figure 4.10. Device Page (zProperties)

4.2.3. Managing Device Templates

The Templates page shows all of the performance templates bound by name to this device or group of devices. To access templates, open the Device table menu, and then select More > Templates.

| Name | Definition Path | Description | Copy |
|------------|--|--|-------------------|
| Device | /Devices/Server | Net-SNMP template for late vintage unix device. Has CPU threshold. | Create Local Copy |
| DnsMonitor | /Devices | | Create Local Copy |
| test | /Devices/Server/Linux/devices /build.zenoss.loc | | Remove Local Copy |

Figure 4.11. Device Page (Templates)

For detailed information about performance templates, go to the section titled "Performance Monitoring."

4.2.4. Determining Device Administration

Use the Device Administration page to define commands and specify who holds administration capabilities for the device.

To access the Administration page, open the Device page menu, and then select More > Administration.

The screenshot displays the Zenoss Device Administration interface for a Linux server. The left sidebar contains navigation links for Main Views, Classes, Events, Devices, Services, Processes, Products, Browse By, and Management. The main content area is divided into three sections: Define Commands, Maintenance Windows, and Administrators. The Define Commands section shows a list of commands with checkboxes for selection. The Maintenance Windows section shows a table with columns for Name, Start, Duration, Repeat, Start State, Stop State, and Enabled?. The Administrators section shows a table with columns for Name, Role, Level, Email, and Pager, listing users 'brad' and 'zruiser'.

Figure 4.12. Device Page (Administration)

If you are using Zenoss Core, the Administrators area is informational only. Use this area to define administrators for this device, and to specify their assigned roles.

4.2.5. Clearing Heartbeats

If you have configured a device to send a recurring event that you have mapped to a heartbeat class in Zenoss, you can clear stale heartbeat events.

To clear the heartbeats associated with a device:

1. Navigate to the device.
2. From the Device page menu, select Manage > Clear Heartbeats.

Zenoss moves the heartbeats for the device to event history. The Edit tab for the ZenEventManager appears. Optionally make changes, and then click **Save**.

4.2.6. Pushing Configuration Changes to Zenoss

When you make a configuration change in Zenoss, the change is automatically propagated to all the remote collectors. If you think that your change has not been propagated automatically, then you can manually force a configuration "push" to the collectors.

To push configuration changes:

1. Navigate to the device.
2. From the Device page menu, select Manage > Push Changes.

A status message appears at the upper right of the page, confirming that changes have been pushed to the collectors.

4.2.7. Locking Device Configuration

You can lock a device's configuration to prevent changes from being overwritten when remodeling the device. Two levels of locking are available. You can lock the configuration from deletion and updates, or solely from deletion.

- i** Device locking prevents changes and deletion due to remodeling. It does not prevent manual changes and deletion.

To lock a device configuration:

1. Navigate to the device.
2. From the Device page menu, select Manage > Lock.

The Edit Lock dialog appears.

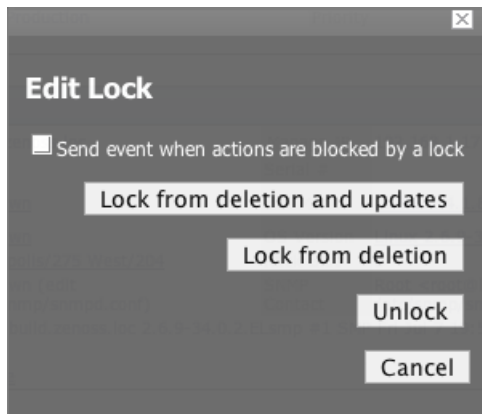


Figure 4.13. Edit (Configuration) Lock Dialog

3. To send events when actions are blocked by a lock action, select the "Send event..." option.
4. Select the type of lock you want to implement, or select Unlock to unlock the device configuration currently set.

The lock or unlock action is implemented on the device.

4.2.8. Renaming a Device

Because Zenoss uses the manage IP to monitor a device, the device name may be different than its fully qualified domain name (FQDN). The device name must always be unique within Zenoss.

To rename a device in Zenoss:

1. Navigate to the device.
2. From the Device page menu, select Manage > Rename Device.

The Rename Device dialog appears.

3. In the ID field, enter the new name for the device.
4. Click **OK**.

The device is renamed in Zenoss.

4.2.9. Remodeling a Device

Remodeling forces Zenoss to re-collect all configuration information associated with a device. Normally, Zenoss models devices every 720 minutes; however, if you want to remodel a device immediately, follow these steps:

1. Navigate to the device.
2. From the Device page menu, select Manage > Model Device.

The device is remodeled, and the remodeling status page appears.

4.2.10. Resetting the Device Manage IP Address

You might want to reset the manage IP address if the IP address of a device has changed and you want to maintain the historical data at the original IP address. To reset the manage IP address of a device in Zenoss:

1. Navigate to the device.
2. From the Device page menu, select Manage > Reset IP.

The Reset IP dialog appears.

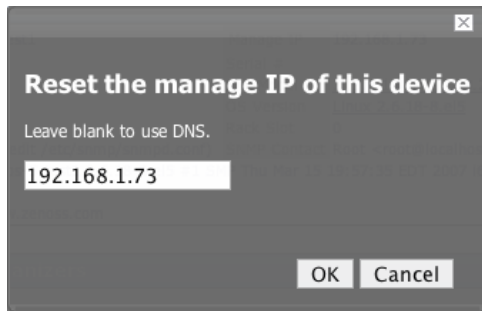


Figure 4.14. Reset IP Dialog

3. Enter the new IP address for the device, or leave the field blank to allow the IP address to be set by DNS.
4. Click **OK**.

The IP address for the device is reset.

4.2.11. Resetting the Device Community

If Zenoss is unable to monitor a device because its SNMP community has changed, you can re-discover the device community by using the list of community strings defined in the zSnmpCommunity zProperty.

1. Navigate to the device.
2. From the Device page menu, select Manage > Reset Community.

The community for the device is reset.

4.2.12. Selecting Device Collector Plugins

The Collector Plugins page lists all of the plugins available for a device.

To access collector plugins:

1. Navigate to the device.
2. From the Device page menu, select More > Collector Plugins.

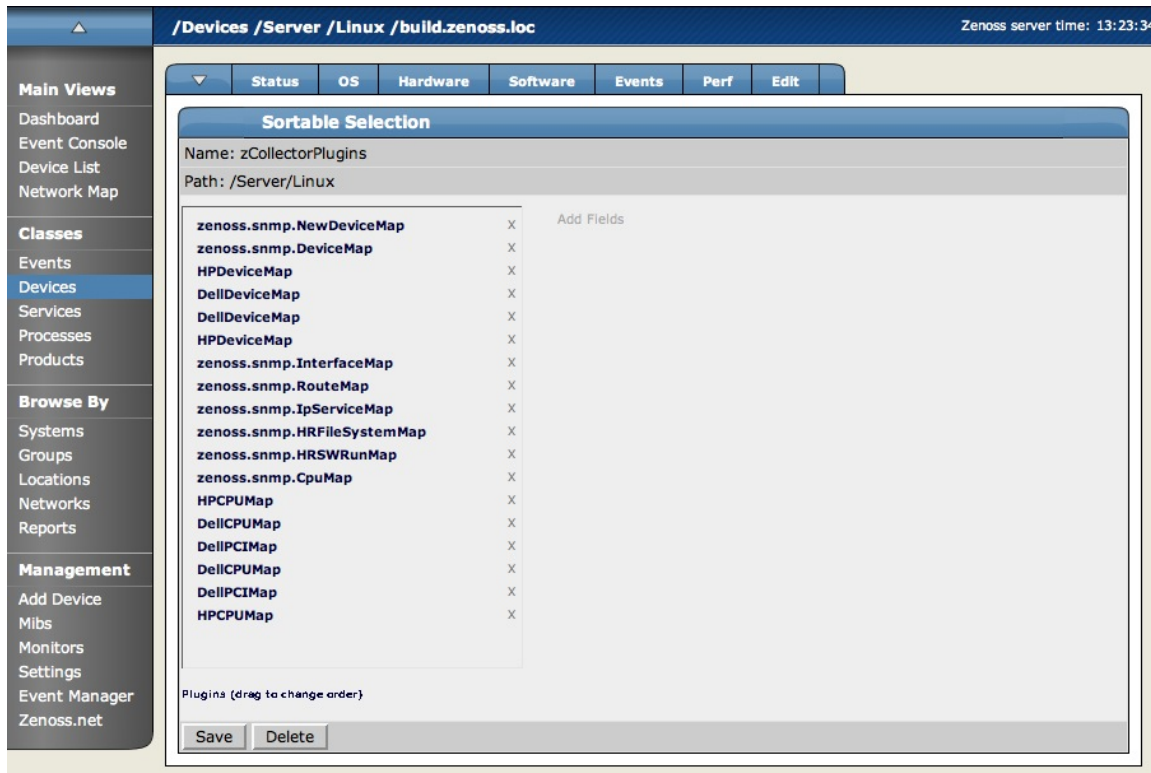


Figure 4.15. Device Page (Collector Plugins)

4.2.13. Deleting a Device from Zenoss

To delete a device from Zenoss:

1. Navigate to the device.
2. From the Device page menu, select Manage > Delete Device.

The Delete Device dialog appears.

3. Click **OK** to confirm deletion.

4.2.14. Managing Multiple Devices from the Device List

You can perform many management tasks for more than one device at a time. Make selections from the Device list page menu to manage multiple devices. Select devices in the list (check the box next to each one you want to select), and then select an option from the menu.

When managing multiple devices, you can:

- Move devices to new classes
- Assign devices to groups, systems, and locations
- Assign monitors for collecting from selected devices
- Remove devices from Zenoss
- Provide configuration locks for devices

4.2.15. Dumping and Loading Devices Using XML

Zenoss allows you to export a list of your devices to an XML file to import into another Zenoss instance. From the command line, use the command:

```
zendevicedump -o mydevicelist.xml
```

This command writes the names of your devices (including their device classes, groups, systems) to a file named `mydevicelist.xml`.

To load these devices into another Zenoss instance (or reload them into the same instance), while in the Zenoss instance where you want the devices to be discovered, run the command:

```
zendeviceload -i mydevicelist.xml
```

Zenoss attempts to discover each of the devices in the XML file.

Chapter 5. Properties and Templates

Read this chapter to learn more about Zenoss properties (*zProperties*) and monitoring templates, and how they control system monitoring.

5.1. zProperties

zProperties are individual values that exist on the major configuration organizers in the system:

- **Device classes**

Device class zProperties control the way monitoring is gathered from devices after they have been added to the system.

- **Networks**

Network zProperties control the way Zenoss performs auto-discovery.

- **Events**

Event zProperties control the rules that process events as they are added to the system.

- **Services and processes**

Service and process zProperties describe how operating system components are monitored by the Zenoss monitoring daemons.

zProperties can be configured for:

- All items
- Single devices
- Multiple devices in the device hierarchy

zProperties settings can be added to any ZenPacks you create, allowing you to add customized zProperties to the system when you add ZenPacks.

5.1.1. zProperties Inheritance and Override

The following diagram illustrates a portion of Zenoss' standard *device class* hierarchy. (A device class is a special type of organizer used to manage how the system models and monitors devices.)

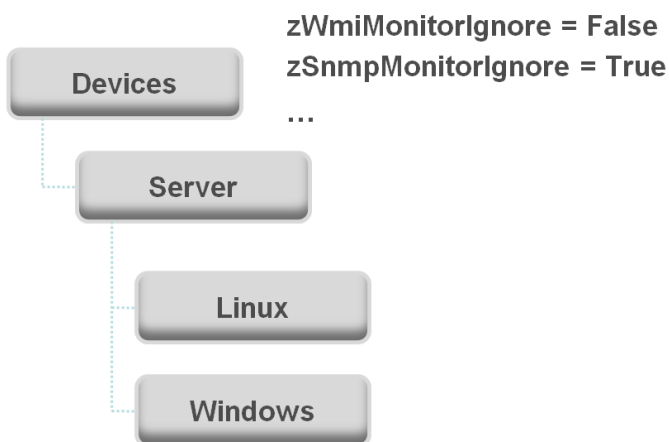


Figure 5.1. Device Class Hierarchy

At the root of the device hierarchy is the Devices object. All device class zProperties are defined here, and their values are the default values for the entire hierarchy.

The illustration further shows two defined zProperties:

- **zWmiMonitorIgnore** - Turns off all daemons that use WMI. By default its value at the root of the hierarchy is False.
- **zSNMPMonitorIgnore** - Turns off all daemons that use SNMP. By default, its value at the root of the hierarchy is True.

Through *inheritance*, properties defined at the root of the hierarchy apply to all objects beneath that node. So, at the /Devices/Server/Linux level of the device class hierarchy, the value of these two properties is the same as at /Devices, even though the property is not set explicitly at /Devices/Server/Linux. Inheritance simplifies system configuration, because default values set at the root level apply to all devices irrespective of their device class.

To further customize the system, you can change a specific zProperty further down the hierarchy without having to change the definitions of other zProperties. As shown in the following illustration, the value of zWmiMonitorIgnore is changed so that WMI monitoring is performed at the /Devices/Server/Windows level.

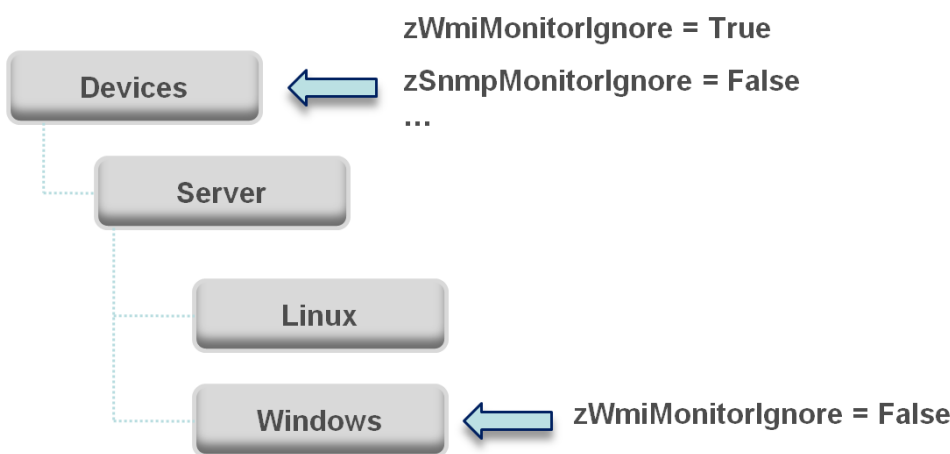


Figure 5.2. Device Class Hierarchy - Locally Defined Value (Override)

This locally defined value for zWmiMonitorIgnore *overrides* the value set at the root of the hierarchy. No other properties at this level are affected by this local change; they continue to inherit the value set at the root.

zProperties allow you to configure the system at a very granular level, down to a particular device. For example, in the following illustration, the device named dev.zenoss.com has the value of SNMP community set to private. This overrides the root value (public).

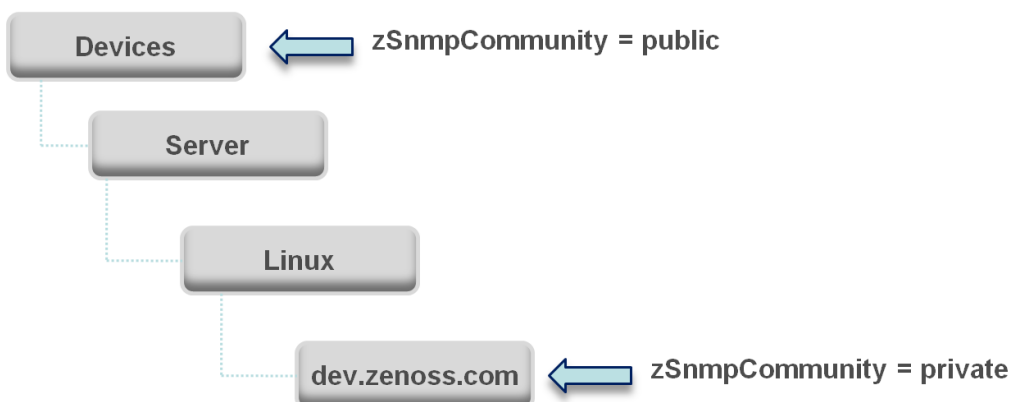


Figure 5.3. Device Class Hierarchy - Value Set on Device

If you change the SNMP Community value of dev.zenoss.com to public, it matches the value set at the root, but is still explicitly defined. Only if you remove the locally defined property does it again inherit the value of the property set at the root.

5.1.1.1. User Interface View

This section further illustrates the characteristics of zProperties from the user interface perspective. The following screen shows zProperties defined at the root level.

The screenshot shows the Zenoss user interface for zProperties Configuration at the root level. The table lists various properties with their values, types, and paths. The Zenoss server time is 11:42:34.

| Property | Value | Type | Path |
|-------------------------|---------------------|---------|------|
| zAxlPassword | | string | / |
| zAxlUsername | administrator | string | / |
| zCollectorClientTimeout | 180 | int | / |
| zCollectorDecoding | latin-1 | string | / |
| zCollectorLogChanges | True | boolean | / |
| zCollectorPlugins | Edit | lines | / |
| zCommandCommandTimeout | 15.0 | float | / |
| zCommandCycleTime | 60 | int | / |
| zCommandExistenceTest | test -f %s | string | / |
| zCommandLoginTimeout | 10.0 | float | / |
| zCommandLoginTries | 1 | int | / |
| zCommandPassword | | string | / |
| zCommandPath | /opt/zenoss/libexec | string | / |
| zCommandPort | 22 | int | / |
| zCommandProtocol | ssh | string | / |

Figure 5.4. Defined zProperties - Root Level

As shown in the previous screen, the zCollectorClientTimeout zProperty has a default value of 180. In the next screen, the value has been set to 170 at /Devices/Server/Linux, overriding the default value at this node of the hierarchy.

The screenshot shows the Zenoss user interface for zProperties Configuration at the /Devices/Server/Linux node. The table lists various properties with their values, types, and paths. The Zenoss server time is 11:44:06. A callout box points to the zCollectorClientTimeout row, stating: "Shows the node at which the default value is overridden."

| Property | Value | Type | Path |
|-------------------------|---------------|---------|---------------|
| zAxlPassword | | string | / |
| zAxlUsername | administrator | string | / |
| zCollectorClientTimeout | 170 | int | /Server/Linux |
| zCollectorDecoding | latin-1 | string | / |
| zCollectorLogChanges | True | boolean | / |
| zCollectorPlugins | Edit | lines | /Server/Linux |
| zCommandCommandTimeout | 15.1 | float | /Server |
| zCommandCycleTime | 60 | int | / |
| zCommandExistenceTest | test -f %s | string | / |
| zCommandLoginTimeout | 10.0 | float | / |
| zCommandLoginTries | 1 | int | / |

Figure 5.5. zCollectorClientTimeout zProperty - Local Value Set

To then remove the override and once again inherit the value from the root of the hierarchy, go to the Delete Local Property area of the zProperties page.

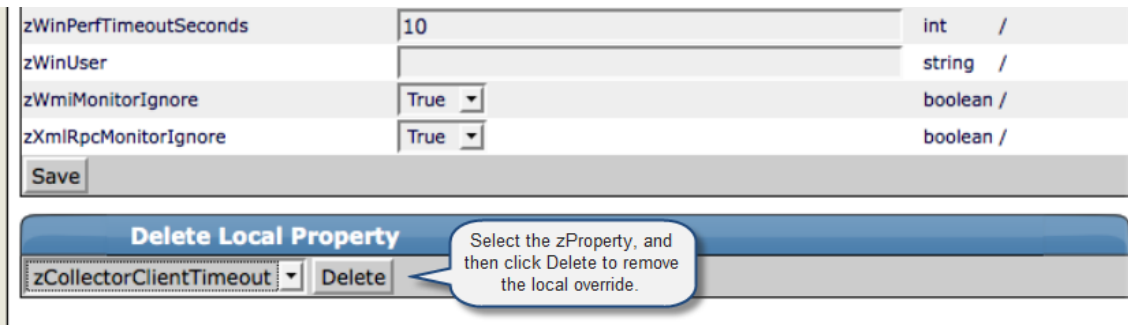


Figure 5.6. Delete Local Property

5.1.2. zProperty Types

zProperties can be one of these types:

- **String** - Text value that can be ASCII or Latin-1 encoded
- **Integer** - Whole number
- **Float** - Number that can have a decimal value
- **Boolean** - True or false
- **Lines** - List of values separated by a return. Zenoss stores these as an array.

5.1.3. Event zProperties

To access Event zProperties, from the left navigation menu, select Event and click the zProperties tab. You can also set zProperties for any event class in the events hierarchy by navigating to the level in the event hierarchy where you want to set the zProperty and clicking the zProperties tab.

| Property Name | Property Type | Description |
|--------------------|---------------|--|
| zEventAction | string | Specifies the location in which an event will be stored. Possible values are: status, history and drop. Default is status, meaning the event will be an “active” event. History sends the event directly to the history table. Drop tells the system to discard the event. |
| zEventClearClasses | lines | Lists classes that a clear event should clear (in addition to its own class). |
| zEventSeverity | int | Overrides the severity value of an event. Possible values are 0 – 5. |

Table 5.1. Event zProperties

5.1.4. Device zProperties

To access Device zProperties, from the left navigational menu, select Devices and then click the zProperties tab. You can also select any of the Device Classes and the zProperties tab to set zProperties anywhere in the Device hierarchy. To set zProperties for an individual Device, navigate to that device, open the page menu, select the More option and then the zProperties tab.

| Property Name | Property Type | Description |
|---------------------------|---------------|--|
| zCollectorClientTimeout | int | Allows you to set the timeout time of the collector client in seconds |
| zCollectorDecoding | string | Converts incoming characters to Unicode. |
| zCollectorLogChanges | Boolean | Indicates whether to log changes. |
| zCollectorPlugins | lines | Links to tall collector plugins for this device. |
| zCommandCommandTimeout | float | Specifies the time to wait for a command to complete. |
| zCommandCycleTime | int | Specifies the cycle time you use when executing zCom- mands for this device or organizer. |
| zCommandExistenceTest | string | *** |
| zCommandLoginTimeout | float | Specifies the time to wait for a login prompt. |
| zCommandLoginTries | int | Sets the number of times to attempt login. |
| zCommandPassword | string | Specifies the password to use when performing command login. |
| zCommandPath | string | Sets the default path where ZenCommand plug-ins are in- stalled on the local Zenoss box (or on a remote box where SSH is used to run the command). |
| zCommandPort | int | Specifies the port to connect to when performing command collection. |
| zCommandProtocol | string | Establishes the protocol to use when performing command collection. Possible values are SSH and telnet. |
| zCommandSearchPath | lines | Sets the path to search for any commands. |
| zCommandUsername | string | Specifies the user name to use when performing command collection. |
| zDeviceTemplates | lines | Sets the templates associated with this device. Linked by name. |
| zFileSystemMapIgnoreNames | string | Sets a regular expression of file system names to ignore. |
| zFileSystemMapIgnoreTypes | lines | Do not use. |
| zIcon | lines | Specifies the icon to represent the device wherever device Icon is shown, such as on the network map and device sta- tus page. Most devices, including Windows servers, Linux servers, and routers, have images set by default. |
| zIfDescription | Boolean | Shows the interface description field in the interface list. |
| zInterfaceMapIgnoreNames | string | Filters out interfaces that should not be discovered. |
| zInterfaceMapIgnoreTypes | string | Filters out interface maps that should not be discovered. |
| zIpServiceMapMaxPort | int | Specifies the highest port to scan. The default is 1024. |
| zKeyPath | lines | Sets the path to the key to access a device. |

Table 5.2. Device zProperties

| Property Name | Property Type | Description |
|------------------------------|---------------|---|
| zLinks | text | Specifies a place to enter any links associated with the device. |
| zLocalInterfaceNames | string | Regular expression that uses interface name to determine whether the IP addresses on an interface should be incorporated into Zenoss' network map. For instance, a loopback interface "lo" might be excluded. |
| zLocalIpAddresses | int | Specifies IP addresses that should be excluded from the network map (for example. 127.x addresses). If you have addresses that you reuse for connections between clustered machines they might be added here as well. |
| zMaxOIDPerRequest | int | Sets the maximum number of OIDs to be sent by Zenoss' SNMP collection daemons when querying information. Some devices have small buffers for handling this information so the number should be lowered. |
| zPingInterfaceDescription | string | A catalog query string that will find interfaces to be pinged by description. |
| zPingInterfaceName | string | A catalog query string that will find interfaces to be pinged by name. |
| zPingMonitorIgnore | Boolean | Whether or not to ping the device. |
| zProdStateThreshold | int | Production state threshold at which Zenoss will begin to monitor a device. Default of 500 equals Pre-Productions. |
| zPythonClass | string | DO NOT USE |
| zRouteMapCollectOnlyIndirect | Boolean | Only collect routes that are directly connected to the device. |
| zRouteMapCollectOnlyLocal | Boolean | Only collect local routes. (These usually are manually configured rather than learned through a routing protocol.) |
| zSnmpAuthPassword | string | The shared private key used for authentication. Must be at least 8 characters long. |
| zSnmpAuthType | string | Use either "MD5" or "SHA" signatures to authenticate SNMP requests |
| zSnmpCommunities | lines | Array of SNMP community strings that the ZenModeler will try to use when collecting SNMP information. |
| zSnmpCommunity | string | Community to be used when collecting SNMP information. If it is different than what is found by ZenModeler, it will be set on the modeled device. |
| zSnmpMonitorIgnore | Boolean | Whether or not to ignore monitoring SNMP on a device. |
| zSnmpPort | int | Port that the SNMP agent listens on. |
| zSnmpPrivPassword | string | The shared private key used for encrypting SNMP requests. Must be at least 8 characters long. |
| zSnmpPrivType | string | "DES" or "AES" cryptographic algorithms. |
| zSnmpSecurityName | string | The Security Name (user) to use when making SNMPv3 requests. |
| zSnmpTimeout | float | Timeout time in seconds for an SNMP request |
| zSnmpTries | int | Amount of tries to collect SNMP data |
| zSnmpVer | string | SNMP version used. Valid values are v1, v2, v3. |

Table 5.3. Device zProperties

| Property Name | Property Type | Description |
|-----------------------------|---------------|--|
| zStatusConnectTimeout | | The amount of time that the zenstatus daemon should wait before marking an IP service down. |
| zSysedgeDiskMapIgnore-Names | | Currently unused. |
| zTelnetEnable | Boolean | When logging into a Cisco device issue the enable command to enable access during command collection. |
| zTelnetEnableRegex | string | Regular expression to match the enable prompt. |
| zTelnetLoginRegex | string | Regular expression to match the login prompt. |
| zTelnetPasswordRegex | string | Regular expression to match the password prompt. |
| zTelnetPromptTimeout | float | Time to wait for the telnet prompt to return. |
| zTelnetSuccessRegexList | lines | List of regular expressions to match the command prompt. |
| zTelnetTermLength | Boolean | On a Cisco device, set term length to Zero. |
| zWinEventlog | Boolean | Whether or not to send the log. |
| zWinEventlogMinSeverity | int | Sets minimum severity to collect from the win event log. The higher the number, the lower the severity. (1 is most severe; 5 is least severe.) |
| zWinPassword | string | The password used to remotely login if it is a Windows machine. |
| zWinServices | | |
| zWinUser | string | User name used to remotely login if it is a Windows machine. |
| zWmiMonitorIgnore | Boolean | Use to turn on or off all WMI monitoring. |
| zXmlRpcMonitorIgnore | Boolean | Use to turn on or off all XML/RPC monitoring. |

Table 5.4. Device zProperties

5.1.5. Service zProperties

To access Service zProperties, from the left navigation menu, select Services, and then click the zProperties tab. You can also access the Service zProperties tab anywhere in the services hierarchy by going to the level where you want to set the zProperty and clicking the zProperties tab.

| Property Name | Property Type | Description |
|---------------------|---------------|---|
| zFailSeverity | int | Determines what severity to send for the specified service. |
| zHideFieldsFromList | lines | Fields to hide from Services instance list |
| zMonitor | Boolean | Tells whether or not to monitor a service. |

Table 5.5. Service zProperties

5.1.6. Process zProperties

To access Process zProperties, from the left navigation menu, select Processes, and then click the zProperties tab. You can also access the Process zProperties tab anywhere in the processes hierarchy by going to the level where you want to set the zProperty and clicking the zProperties tab.

| Property Name | Property Type | Description |
|-----------------|---------------|---|
| zAlertOnRestart | Boolean | Determines whether or not to send an event if the specified process is restarted. |
| zCountProcs | int | Determines the number of instances of the process that are running. |
| zFailSeverity | int | Determines what severity to send for the specified process. |
| zMonitor | Boolean | Tells whether or not to monitor a process. |

Table 5.6. Process zProperties

5.1.7. Network zProperties

To access Network zProperties, from the left navigation menu, select Networks, and then click the zProperties tab. You can also access the zProperties tab for any sub-network that exists within the Networks page.

| Property Name | Property Type | Description |
|---------------------|---------------|---|
| zAutoDiscover | Boolean | Should zendisc perform auto-discovery on this network |
| zDefaultNetworkTree | lines | List of netmask numbers to use when creating network containers. Default is 24, 32 which will make /24 networks at the top level of the networks tree if a network is smaller than /24. |
| zPingFailThresh | int | Number of pings to sent without being returned before Zendisc removes the device. |

Table 5.7. Network zProperties

5.1.8. Manufacturer zProperties

| Property Name | Property Type | Description |
|---------------|---------------|--------------------------|
| zDeviceClass | string | Reserved for future use. |
| zDeviceGroup | string | Reserved for future use. |
| zSystem | string | Reserved for future use. |

Table 5.8. Network zProperties

5.2. Templates

Zenoss stores performance configuration data in RRDTemplates (generally referred to as *templates*). Templates contain other objects that define where and how to obtain performance data, thresholds for that data, and data graphs.

You can define a template anywhere in the device class hierarchy, or on an individual device.

Templates are divided among three types: device, component, and interface.

5.2.1. Template Binding

The determination of which templates apply to what objects is called *binding*. Templates are bound in different ways, depending on the objects to which they are bound.

5.2.1.1. Device Templates

Device templates are applied to devices, one to each device. Zenoss employs a single rule to bind device templates to devices: the value of the zDeviceTemplates property. For most device classes, this is "Device."


Common device templates are:

- Device
- MySQL, Apache
- Active Directory, MExchangeIS, MSSQLServer, IIS

For the Server/Linux/MySQL device class, the `zDeviceTemplates` property might contain, for example, "Device" and "MySQL." Zenoss would collect CPU and memory information by using the Device template, and MySQL-specific metrics by using the MySQL template.

5.2.1.2. Component Templates

Component templates are named exactly according to the name of the underlying class that represents a component. For example, the `FileSystem` template is applied to file systems. Component templates can be applied multiple times to each device, depending on how many of the device's components match the template. `zProperties` do not control the application of component templates.

 Component templates should not be manually bound.

Common component templates are:

- `FileSystem`, `HardDisk`, `IPService`, `OSProcess`, `WinService`
- Fan, `PowerSupply`, `TemperatureSensor`
- `LTMVirtualServer`, `VPNTunnel`

5.2.1.3. Interface Templates

Interface templates are applied to network interfaces by using a special type of binding. Instead of using the name of the underlying class, Zenoss looks for a template with the same name as the interface type. You can find this type in the details information for any network interface (from the OS tab of its containing device).

If Zenoss cannot locate a template that matches the interface type, then it uses the `ethernetCsmacd` template.

5.2.1.4. Defining Templates in the Device Hierarchy

You add a new device at `/Devices/Server/Linux/Example1Server`. You have not edited the value of its `zDeviceTemplates` property, so it inherits the value of "Device" from the root device class (`/Devices`). Zenoss looks to see if there is a template named `Device` defined on `Example1Server` itself. There is not, so it checks `/Devices/Server/Linux`. There is a template named `Device` defined for that device class, so that template is used for `Example1Server`. (There also is a template named `Device` defined at the root level (`/Devices`), but Zenoss does not use this one because the template at `/Devices/Server/Linux` overrides it.)

5.2.1.5. Applying Templates to Multiple Areas in the Device Hierarchy

You want to perform specific monitoring of servers running a certain Web application, but those servers are spread across several different device classes. You create a template at `/Devices` called `WebApplication` with the appropriate data sources, thresholds and graphs. You then append the name "WebApplication" to the `zDeviceTemplates` `zProperty` for the device classes, the individual devices running this Web application, or both.

Chapter 6. Core Monitoring

Read the following sections for more information about basic and advanced Zenoss monitoring:

- Availability Monitoring
- Performance Monitoring
- Monitoring Using ZenCommand
- SNMP Monitoring
- Monitoring Device Remotely Via SSH
- Monitoring Windows Devices

6.1. Availability Monitoring

The availability monitoring system within Zenoss provides active testing of the IT Infrastructure. The system currently consists of ZenPing, Zenoss' Layer-3 aware topology-monitoring daemon, and ZenStatus, a TCP status tester.

ZenPing is configured automatically. ZenPing does the high-performance asynchronous testing of the ICMP status. The most important element of this daemon is that Zenoss has built a complete model of the your routing system. If there are gaps in Zenoss' routing model, the power of ZenPing's topology monitoring will not be available. If there are these gaps, this issue can be seen in the `zenping.log` file.

Zenmodeler goes out and discovers the routes to each device in the Zenoss network. Zenoss tries not to use Internet routing tables and prefers to rely on Zenmodeler to discover the relationships on its own and create its own network map.

Basically if any known route is broken, there will only be one ping event that is generated by the outage. Any additional outages beyond that will only flag that device and the next time a ping sweep occurs the errors beyond the known router will not occur.

This monitoring model breaks down if the routers do not share their routing tables and interface information.

6.1.1. Controlling the Ping Cycle Time

Follow these steps to set up the ping cycle time.

1. From the left Navigation menu, select Monitors and select the Status monitor localhost and click the localhost link.
2. Notice the list of machines being pinged by this monitor.
3. In the "Edit" tab, change "Cycle Interval" to the desired interval.
4. On the next configuration cycle, the ping monitor will ping at the interval you set.

6.1.2. Using the Predefined /Ping Device Class

The /Ping device class is an example of a configuration for devices that should only be monitored for availability. Zenoss will not gather performance data for devices placed under this class; it will only ping them. You can use it as a reference for your own configuration; or, if you have a device that you want be monitored for availability alone, you can place it under this class.

6.1.3. Monitoring TCP Services

Use the Service menu to manage and monitor services that are running on your networks. To access the Services pages, from the left Navigational menu, choose Services. The Services Overview appears.

The Services overview shows the folders and sub-folders and lists all of the services that have been added to the system to monitor.

6.1.3.1. ZenStatus

ZenStatus performs monitoring of TCP services. It is configured by turning on monitoring of a service under the “Services” root on the Navigation Toolbar. Service monitoring can be turned on a service class but this can be overridden on any service instance. For example, “SMTP” will be monitored by default but it may not be a critical service on all boxes. If this is the case, it may be removed on specific devices. Also, if the service is configured to only listen on localhost (127.0.0.1) the service will not be monitored.

6.1.3.2. Adding a Service to Monitor

To add a service to monitor:

1. From the Services Overview page, in the Services text field, enter the name of the service you want to monitor.
2. Click the Add button.

The service appears in the Services list.

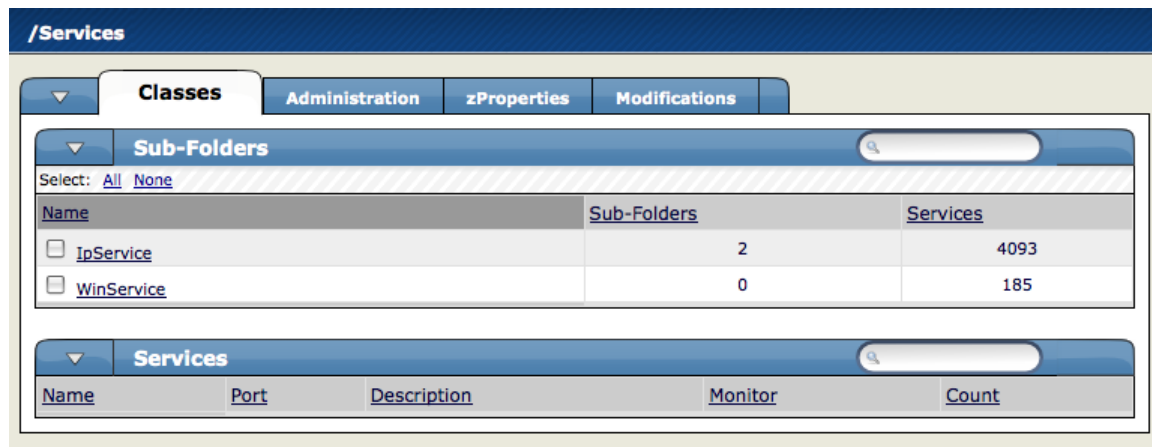


Figure 6.1. Services List - Classes tab

3. To set monitoring to True, click the Edit tab and set the Monitor pop-up to True. The service is now being monitored.

6.1.3.3. Monitoring Status Service Status Information

To view the status information associated with a given service, select the service from the Services list in the Services Overview page.

The Individual Service Status tab appears.

/Services /IpService /tcp_00225

▼ **Status** Edit Administration zProperties Modifications

Service Class

| | | | |
|--------------|-----------|--------------|-------|
| Name | tcp_00225 | Monitor | False |
| Port | 225 | Description | |
| Send String | | Expect Regex | |
| Service Keys | tcp_00225 | | |

Service Instances

| Device | Name | Monitor | Status |
|-----------------------------------|-----------|---------|--------|
| darwin.zenoss.com | tcp_00225 | False | None |
| pub.zenoss.com | tcp_00225 | False | None |
| dev.zenoss.org | tcp_00225 | False | None |

Figure 6.2. Individual Service Status Tab

This tab shows you a summary of the details associated with this service. You can see the name of the service, whether its monitored or not, a Description, any associated Service Keys and a List of Devices where the service is currently running. To change any of this information, click the Edit tab.

6.1.3.4. Editing Service Information

To edit the information that appears on the Individual Service Status tab, from the Individual Services page, click the Edit tab.

/Services /IpService /tcp6_00022

▼ **Status** **Edit** Administration zProperties Modifications

State at time: 2008/06/18 11:35:21

| | | | |
|--------------|------------|------|----|
| Monitor | False ▼ | | |
| Name | tcp6_00022 | Port | 22 |
| Description | | | |
| Send String | | | |
| Expect Regex | | | |
| Service Keys | tcp6_00022 | | |

Save

Figure 6.3. Individual Service - Edit Tab

From this screen, use the Monitor pop-up to select True to monitor the service and False to not monitor the service. You can also add any associated Service Keys or enter a brief Description.

6.1.3.5. Configuring Service zProperties

You can configure zProperties either for all services, for an individual service or for any services that fall further down in the service hierarchy tree. To configure them for all services click the zProperties tab from the Ser-

vice Overview tab. To configure zProperties for an individual service, click on the service name in the Service Overview and then click the zProperties tab for that service. The Service zProperties Tab appears.

zProperties Configuration

| Property | Value | Type | Path |
|---------------------|----------|---------|------|
| zFailSeverity | Critical | int | / |
| zHideFieldsFromList | | lines | / |
| zMonitor | False | boolean | / |

Save

Delete Local Property

▼ Delete

Figure 6.4. Individual Service - zProperties Tab

You can configure the following zProperties for an individual service from this tab:

- zFailSeverity
- zHideFieldsFromList
- zMonitor

For more information about the Service zProperties, see the chapter titled "Properties and Templates."

6.1.3.6. Using the Predefined /Server/Scan Device Class

The /Server/Scan device class is an example configuration for monitoring TCP services on devices using a port scan. You can use this device class as a reference for your own configuration; or, if you have a device that you want to be monitored for service availability alone, you can place it under this device class. Zenoss will not collect performance data for devices in this class.

6.1.3.7. Monitoring a Service Using a Service Class

This section will show you how to set up monitoring of an IP service across a group of devices using a service class.

1. Navigate to the OS tab for a device you have loaded into the system.

The OS Tab for a device appears.

/Devices /Server /Linux /pub.zenoss.com

▼ **Status** **OS** Hardware Software Events Perf Edit

▼ **Interfaces**

Select: [All](#) [None](#)

| | Name | IP Address | Network | MAC | O | A | Lock |
|--------------------------|------|------------------|--------------|-------------------|---|---|------|
| <input type="checkbox"/> | eth0 | 64.34.170.109/26 | 64.34.170.64 | 00:02:B3:E6:8B:90 | | | |
| <input type="checkbox"/> | lo | 127.0.0.1/8 | | | | | |
| <input type="checkbox"/> | sit0 | | | | | | |

▼ **Win Services** ☒ Monitored

[Caption](#) [StartMode](#) [StartName](#) [Name](#) [Status](#) [Lock](#)

1 of 0 show all Page Size 40 ok

▼ **OS Processes**

| | Class | Name | Restarts | Fail Severity | Status | Lock |
|--------------------------|--------|-----------------|----------|---------------|--------|------|
| <input type="checkbox"/> | /snmpd | /usr/sbin/snmpd | True | Error | | |

▼ **IP Services** ☒ Monitored

[Name](#) [Proto](#) [Port](#) [Ips](#) [Description](#) [Status](#) [Lock](#)

1 of 0 show all Page Size 40 ok

▼ **File Systems**

| | Mount | Total bytes | Used bytes | Free bytes | % Util | Lock |
|--------------------------|----------|-------------|------------|------------|--------|------|
| <input type="checkbox"/> | / | 73.4GB | 61.3GB | 12.1GB | 83 | |
| <input type="checkbox"/> | /backups | 75.5GB | 47.9GB | 27.6GB | 63 | |
| <input type="checkbox"/> | /boot | 98.7MB | 8.3MB | 90.4MB | 8 | |

▼ **Routes**

Figure 6.5. Showing Processes to Monitor

- In the IP Services area, click the link to the service you want to monitor.

The service summary for the service you have selected appears.

The screenshot shows the Zenoss web interface for a service named 'tcp_00080'. The breadcrumb path is '/Devices /Server /Linux /nightlytest.zenoss.loc /os /tcp_00080'. The Zenoss server time is 12:25:57. The 'Status' tab is selected, showing the service's state at 2009/10/08 12:25:26. The service is currently down (indicated by a black dot). The configuration shows it is a 'tcp' service on port '80' with IP address '0.0.0.0'. The 'Monitor' flag is set to 'True', and the 'Fail Severity' is set to 'Critical'. A 'Save' button is at the bottom.

| | |
|------------------|--|
| Status | |
| Service Class | /IpService/Privileged/http |
| Name | tcp_00080 |
| Description | World Wide Web HTTP |
| Protocol | tcp |
| Port | 80 |
| IP Addresses | 0.0.0.0 |
| Check IP Address | 10.175.211.95 ▼ |
| Monitor | True ▼ |
| Send String | |
| Expect Regex | |
| Fail Severity | Critical ▼ |
| Locks | |

Save

Figure 6.6. Service Summary

3. Set the Monitor flag to True to monitor this service for only this machine. You can also set this service to be monitored system-wide.
4. To monitor a service system wide, click the Service Class link. This page will show you where the service is running and whether or not the service is monitored.
5. Click the Edit tab and set Monitored to True. This turns on monitoring for every instance of this service in the system.
6. Click **Save**.
7. Click the Status tab again.

Most of the instances of the Service are now set to green, indicating they are monitored and up. The ones that remain unmonitored indicate that they have this service class set to not monitor at a local level.

6.1.4. Monitoring Processes

Zenoss is able to monitor all processes running on your network. Zenoss process monitoring flow is illustrated by the following diagram:

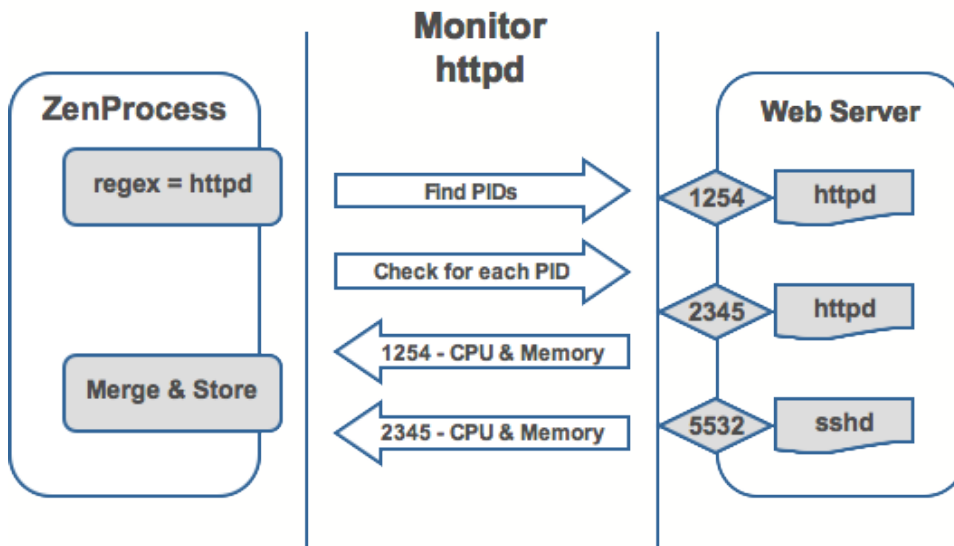


Figure 6.7. Process Monitoring

ZenProcess uses a regular expression match to find PIDs matching the expression to see that these processes are running on the selected device.

6.1.4.1. Adding Processes to Monitor

To add a process to monitor:

1. From the left navigation menu, select Processes.

The Processes page appears.

| Name | Sub-Folders | Processes |
|---------------------------------|-------------|-----------|
| <input type="checkbox"/> Apache | 0 | 1 |
| <input type="checkbox"/> MySQL | 0 | 1 |

| Name | Regex | Monitor | Count |
|--|---|---------|-------|
| <input type="checkbox"/> buildbot | buildbot | True | 6 |
| <input type="checkbox"/> firefox.exe | firefox.exe | True | 0 |
| <input type="checkbox"/> ntpd | ntpd | True | 11 |
| <input type="checkbox"/> snmpd | snmpd | True | 20 |
| <input type="checkbox"/> usr_sbin_snmpd 941949f08c9aeb684a4bb99b5c09480e | usr_sbin_snmpd 941949f08c9aeb684a4bb99b5c09480e | True | 0 |
| <input type="checkbox"/> zenjmx | /ZenJMX | True | 0 |

Figure 6.8. Processes Page

2. From the Processes table menu, select Add Process.

The Add OS Process Dialog Appears.



Figure 6.9. Add OS Process Dialog

3. Enter the regular expression name of the process you want to monitor in the Processes field, and then click **OK**.

The process is added. The Processes window re-appears, showing the process you entered.

Now you are monitoring this process. After the device is remodeled (at the next remodeling interval or manually), it will show every device (occurrence) where this process is running. As such, the process is now being monitored wherever it occurs.

Clicking on a specific process will take you to an interface that shows all instances of that process running across machines that have it monitored. If the process has multiple instances the Zenoss will monitor the sum of CPU and memory utilization of all processes as well as the count of total processes running. However, if the process has only a single instance, CPU utilization and memory usage are graphed for the single process. To perform process monitoring using the `zenprocess` daemon, the device's SNMP agent must show the process information through the HOST- RESOURCES MIB.

6.1.4.2. Configuring Process zProperties

You can configure zProperties for all processes, for an individual process, or for any processes that fall further down in the process hierarchy tree. To configure them for all processes click the zProperties tab from the Process Overview tab. To configure zProperties for an individual process, click on the process name in the Process Overview and then click the zProperties tab for that process. The Process zProperties Tab appears.

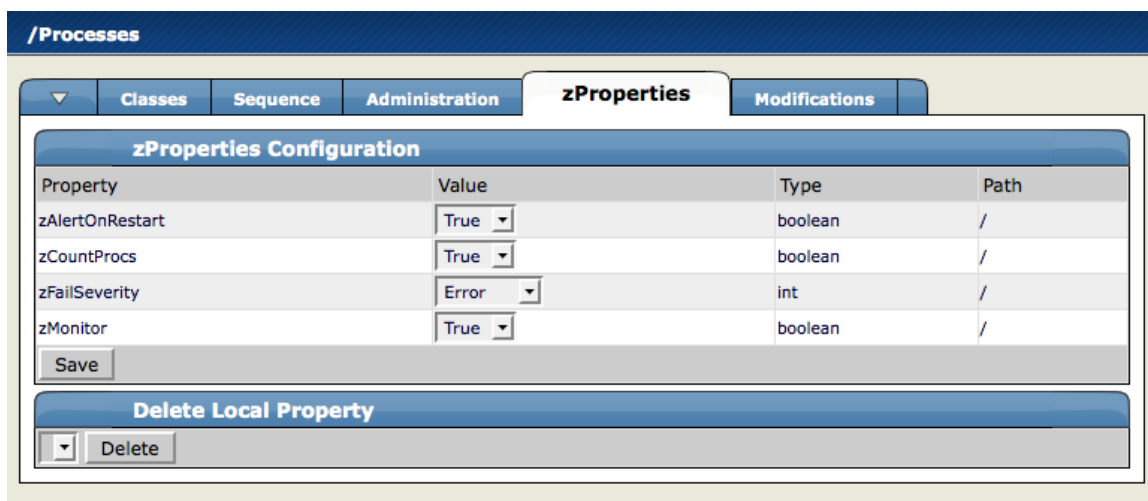


Figure 6.10. Processes zProperties tab

You can configure the following zProperties for either all processes or the selected process from this tab:

- ZAlertOnRestart

- ZCountProcs
- ZFailSeverity
- zMonitor

For more information about the Services zProperties, see the zProperties Appendix.

6.2. Performance Monitoring

Read this chapter to learn about Zenoss performance monitoring and performance templates.

6.2.1. About Performance Monitoring

Zenoss uses several methods to monitor performance metrics of devices and device components. These are:

- **ZenPerfSNMP** - Collects data through SNMP from any device correctly configured for SNMP monitoring.
- **ZenWinPerf** (Zenoss Enterprise only) - ZenPack that allows performance monitoring of Windows servers.
- **ZenCommand** - Logs in to devices (by using telnet or ssh) and runs scripts to collect performance data.
- **Other ZenPacks** - Collect additional performance data. Examples include the ZenJMX ZenPack, which collects data from enterprise Java applications, and the HttpMonitor ZenPack, which checks the availability and responsiveness of Web pages.

Regardless of the monitoring method used, Zenoss stores performance monitoring configuration information in *performance templates*.

6.2.2. Performance Templates

Performance templates determine how Zenoss collects performance data for devices and device components. You can define performance templates for device classes and individual devices.

Templates comprise three types of objects:

- **Data Sources** - Specify the exact data points to collect, and the method to use to collect them.
- **Thresholds** - Define expected bounds for collected data, and specify events to be created in Zenoss if the data does not match those bounds.
- **Graph Definitions** - Describe how to graph the collected data on the device or device components.

Before Zenoss can collect performance data for a device or component, it must determine which templates apply. This process is called *template binding*.

6.2.2.1. Viewing Performance Templates

The Templates page lists all of the performance templates available to a device or device class.

To view available performance templates for a device, select **More > All Templates** from the Devices page menu. To view available performance templates for device classes, click the Templates tab from the Devices area.

The Available Performance Templates page shows the performance templates that are defined for a particular device or device class, and for those defined further up the device class hierarchy. If more than one template of the same name is defined, then only the one to which this device or device class can bind appears in the list.

Click a template in the list to view details about defined data sources, thresholds, and to see graph definition details.

/Devices /Server /Templates /Device

Performance Template

State at time: 2008/06/18 11:10:27

Name: Device

Description: Net-SNMP template for late vintage unix device. Has CPU threshold.

Save

Data Sources

Select: All None

| Name | Source | Source Type | Enabled |
|---|---------------------------|-------------|---------|
| <input type="checkbox"/> laLoadInt5 | 1.3.6.1.4.1.2021.10.1.5.2 | SNMP | True |
| <input type="checkbox"/> memAvailReal | 1.3.6.1.4.1.2021.4.6.0 | SNMP | True |
| <input type="checkbox"/> memAvailSwap | 1.3.6.1.4.1.2021.4.4.0 | SNMP | True |
| <input type="checkbox"/> memBuffer | 1.3.6.1.4.1.2021.4.14.0 | SNMP | True |
| <input type="checkbox"/> memCached | 1.3.6.1.4.1.2021.4.15.0 | SNMP | True |
| <input type="checkbox"/> ssCpuRawIdle | 1.3.6.1.4.1.2021.11.53.0 | SNMP | True |
| <input type="checkbox"/> ssCpuRawSystem | 1.3.6.1.4.1.2021.11.52.0 | SNMP | True |
| <input type="checkbox"/> ssCpuRawUser | 1.3.6.1.4.1.2021.11.50.0 | SNMP | True |
| <input type="checkbox"/> ssCpuRawWait | 1.3.6.1.4.1.2021.11.54.0 | SNMP | True |
| <input type="checkbox"/> sysUpTime | 1.3.6.1.2.1.25.1.1.0 | SNMP | True |

1 of 10 |< < laLoadInt5 > >| show all Page Size 40 ok

Thresholds

| Name | Type | Data Points | Severity | Enabled |
|--|-----------------|---------------------------|----------|---------|
| <input type="checkbox"/> CPU Utilization | MinMaxThreshold | ssCpuRawIdle_ssCpuRawIdle | Warning | True |

Graph Definitions

Select: All None

| Seq | Name | Graph Points | Units | Height | Width |
|-----|---|--|------------|--------|-------|
| 0 | <input type="checkbox"/> Load Average | laLoadInt5 | load | 100 | 500 |
| 0 | <input type="checkbox"/> Load Average 5 min | laLoadInt5 | processes | 100 | 500 |
| 1 | <input type="checkbox"/> CPU Utilization | ssCpuRawSystem, ssCpuRawUser, ssCpuRawWait | percentage | 100 | 500 |
| 2 | <input type="checkbox"/> CPU Idle | CPU Utilization, ssCpuRawIdle | percentage | 100 | 500 |
| 3 | <input type="checkbox"/> Free Swap | memAvailSwap | KBytes | 100 | 500 |
| 4 | <input type="checkbox"/> Free Memory | memAvailReal | bytes | 100 | 500 |

Figure 6.11. Performance Template for Load Average Graph

6.2.3. Template Binding

Before Zenoss can collect performance data for a device or component, it must determine which templates apply. This process is called template binding.

First, Zenoss determines the list of template names that apply to a device or component. For device components, this usually is the meta type of the component (for example, FileSystem, CPU, or HardDisk). For devices, this list is defined by the zDeviceTemplates zProperty.

After defining the list, Zenoss locates templates that match the names on the list. For each name, Zenoss searches the device and then searches the device class hierarchy. Zenoss uses the lowest template in the hierarchy that it can locate with the correct name, ignoring others of the same name that might exist further up the device class hierarchy.

Viewing Templates Available for Binding

To see which templates are available for binding, view the Templates page for any device or device class. This page shows all templates that are defined at this point or higher in the device hierarchy.

Changing Templates Available for Binding


To change which templates are currently bound:

1. Select Bind Templates from the Available Performance Templates table menu.

The Bind Performance Templates dialog appears.

2. Select a performance template (Ctrl-Click to select more than one), and then click **OK**.

The selected performance template appears in the list of available templates.

 Alternatively, you can edit the zDeviceTemplates zProperty (from the zProperties page) to change which templates are bound. You cannot edit the bound name for a device component.

| Name Binding | Definition |
|--------------|---|
| Device | The device object. (These OIDs do not have an snmp index number.) |
| FileSystem | The file system object currently uses the host resources MIB. |
| Interface | Interfaces are bound using their interface type. (For example: ethernetCsmacd.) |
| HardDisk | Hard disk object for I/O stats, such as Windows boxes with Informant MIB. |

6.2.4. Data Sources

Data sources specify which data points to collect and how to collect them. Each performance template comprises one or more data sources. Zenoss provides two built-in data source types: SNMP and COMMAND. (Other data source types are provided through ZenPacks.)

About SNMP Data Sources

SNMP data sources define data to be collected via SNMP by the ZenPerfSNMP daemon. They contain one additional field to specify which SNMP OID to collect. (Many OIDs must end in .0 to work correctly.) Because SNMP data sources specify only one performance metric, they contain a single data point. For more information, see the section titled SNMP Monitoring.

About COMMAND Data Sources

COMMAND data sources specify data to be collected by a shell command that is executed on the Zenoss server or on a monitored device. The ZenCommand daemon processes COMMAND data sources. A COMMAND data source may return one or more performance metrics, and usually has one data point for each metric.

Shell commands used with COMMAND data sources must return data that conforms to the Nagios plug-in output specification. For more information, see the section titled Monitoring Using ZenCommand.

6.2.4.1. Adding a Data Source

To add a data source to a performance template:

1. From the Performance Template page, select Add DataSource from the Data Sources table menu.

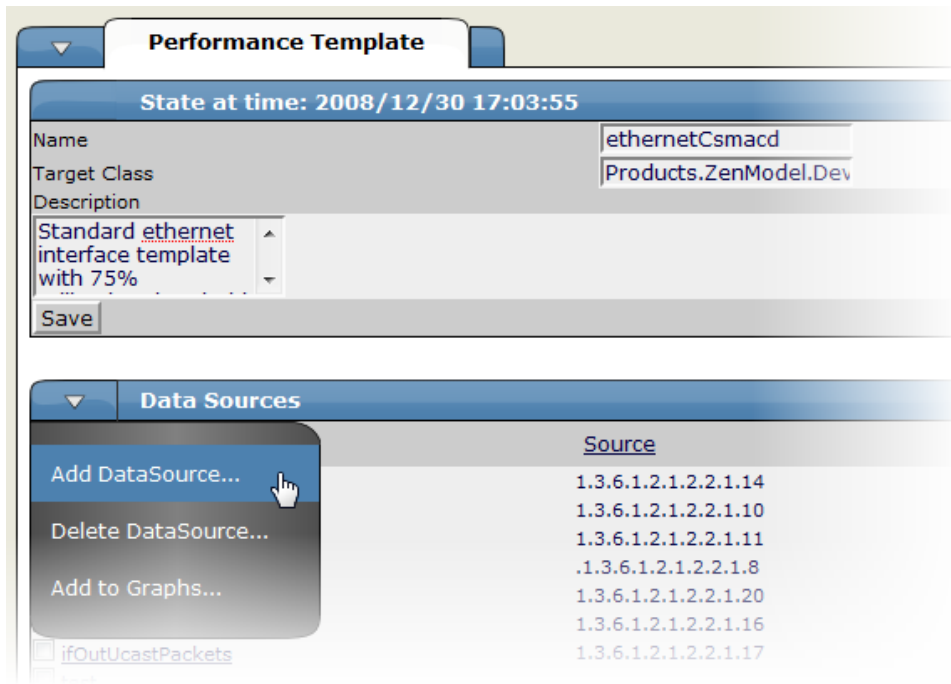


Figure 6.12. Add Data Source

The Data Source page appears.

2. Enter or select values to define the data source.

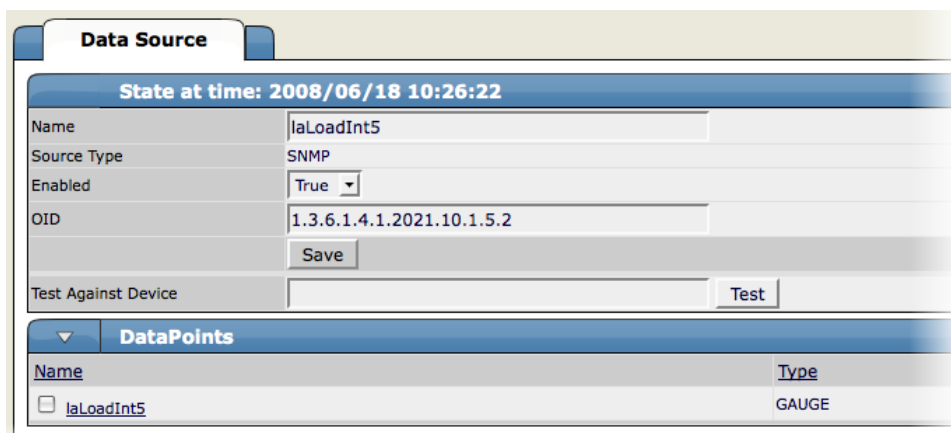


Figure 6.13. SNMP Data Source

6.2.5. Data Points

Data sources can return data for one or more performance metrics. Each metric retrieved by a data source is represented by a data point.


Defining Data Points

You can define data points to data sources with all source types except SNMP and VMware. Because these data source types each rely on a single data point for performance metrics, additional data point definition is not needed.


To add a data point to a data source:

1. Select a data source from the Data Sources area of the Performance Template page.

2. From the DataPoints table menu on the Data Source page, select Add Datapoint.
3. In the Add a New DataPoint dialog, enter a name for the data point, and then click Add.

 For COMMAND data points, the name should be the same as that used by the shell command when returning data.

4. Enter information or make selections to define the data point.
 - Name - Displays the name you entered in the Add a New DataPoint dialog.
 - Type - Specify the RRD data source type to use for storing data for this data point. (Zenoss uses RRD-Tool to store performance data.) Available options are:
 - COUNTER - Saves the rate of change of the value over a step period. This assumes that the value is always increasing (the difference between the current and the previous value is greater than 0). Traffic counters on a router are an ideal candidate for using COUNTER.
 - DERIVED - Same as COUNTER, but additionally allows negative values. If you want to see the rate of change in free disk space on your server, for example, then you might want to select this value.
 - ABSOLUTE - Saves the rate of change, but assumes that the previous value is set to 0. The difference between the current and the previous value is always equal to the current value. Thus, ABSOLUTE stores the current value, divided by the step interval.
 - GAUGE - Does not save the rate of change, but saves the actual value. There are no divisions or calculations. To see memory consumption in a server, for example, you might want to select this value.

 Rather than COUNTER, you may want to define a data point using DERIVED and with a minimum of zero. This creates the same conditions as COUNTER, with one exception. Because COUNTER is a "smart" data type, it can wrap the data when a maximum number of values is reached in the system. An issue can occur when there is a loss of reporting and the system (when looking at COUNTER values) thinks it should wrap the data. This creates an artificial spike in the system and creates statistical anomalies.

- RRDMIN - Enter a value. Any value received that is less than this number is ignored.
- RRDMax - Enter a value. Any value received that is greater than this number is ignored.
- Create CMD - Enter an RRD expression used to create the database for this data point. If you do not enter a value, then Zenoss uses a default applicable to most situations.

For details about the rrdcreate command, go to:

<http://oss.oetiker.ch/rrdtool/doc/rrdcreate.en.html>

5. Click Save to save the data point.

Data Point

State at time: 2008/06/18 10:26:24

Name:

Type:

RRD Min:

RRD Max:

Create Cmd:

Figure 6.14. Define a Data Point

6.2.6. Data Point Aliases

Zenoss performance reports pull information from various data points that represent a metric. The report itself knows which data points it requires, and which modifications are needed, if any, to put the data in its proper units and format.

The addition of a data point requires changing the report.

CPU Utilization Report

*For the Linux data point:
Divide the total CPU idle percentage
by the number of cores, and subtract
from 100.*

*For the Windows data point:
Report the CPU utilization unmodified.*

*Adding ESX requires adding another
rule to fetch it and divide the value by
100, because the utilization is reported
on a scale from 0 to 10000.*

| Device | Utilization |
|---------------------|-------------|
| linux.example.com | 20% |
| windows.example.com | 34% |

Total CPU idle = 320%

linux.example.com
(Linux with 4 cores)

CPU utilization = 34%

windows.example.com
(Windows)

CPU utilization = 4322

esx.example.com
(ESX via VI SDK)

Figure 6.15. CPU Utilization Report

To allow for more flexibility in changes, some Zenoss reports use *data point aliases*. Data point aliases group data points so they can be more easily used for reporting. In addition, if the data points return data in different units, then the plugin can normalize that data into a common unit.

An alias-based report looks up the data points that share a common alias string, and then uses them. This approach allows you to add data points without changing the report.

CPU Utilization Report (using aliases)

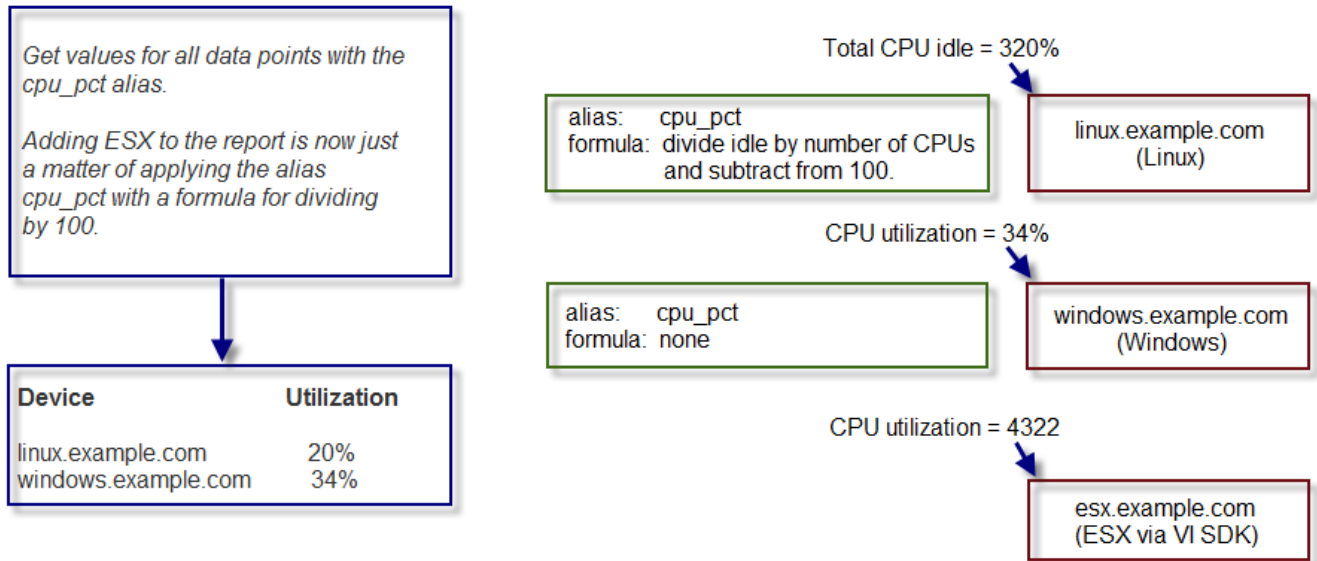


Figure 6.16. Alias-Based CPU Utilization Report

In the simplest cases, data from the target data points are returned in units expected by a report. For cases in which data are not returned in the same units, an alias can use an associated formula at the data point. For example, if a data point returns data in kilobytes, but the report expects data in bytes, then the formula multiplies the value by 1024.

6.2.6.1. Alias Formula Evaluation

Zenoss evaluates the alias formula in three passes.

6.2.6.1.1. Reverse Polish Notation

When complete, the alias formula must resolve to a Reverse Polish Notation (RPN) formula that can be used by RRDtool. For the simple conversion of kilobytes into bytes, the formula is:

```
1024, *
```

For more information on RRDtool and RPN formulas, browse to this site:

http://oss.oetiker.ch/rrdtool/doc/rrdgraph_rpn.en.html

6.2.6.1.2. Using TALES Expressions in Alias Formulas

For cases in which contextual information is needed, the alias formula can contain a TALES expression that has access to the device as context (labeled as "here"). The result of the TALES evaluation should be an RRD formula.

For example, if the desired value is the data point value divided by total memory, the formula is:

```
${here/hw/totalMemory}, /
```

For more information on TALES, refer to the appendix in this guide titled "TALES Expressions," or to the TALES Specification 1.3, at:

<http://wiki.zope.org/ZPT/TALESSpecification13>

6.2.6.1.3. Using Python in Alias Formulas

You also can embed full Python code in an alias formula for execution. The code must construct a string that results in a valid RRD formula. To signal Zenoss to evaluate the formula correctly, it must begin with:

```
__EVAL:
```

Using the same example as in the previous section (division by total memory), the formula is:

```
__EVAL:here.hw.totalMemory + ","/"
```

6.2.6.2. Adding a Data Point Alias

To add a data point alias:

1. Navigate to a data source on a template.
2. Select a data point.
3. From the page menu, select Add DataPoint Alias.

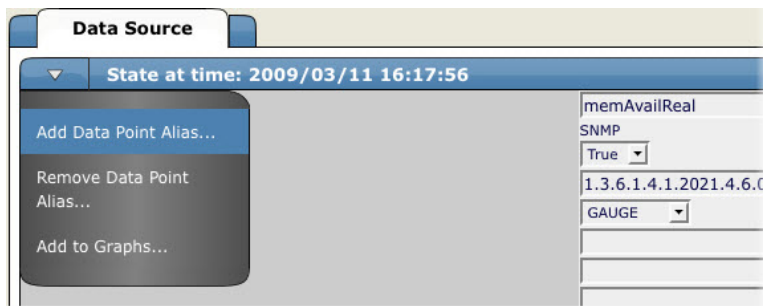


Figure 6.17. Add Data Point Alias

4. In the Add Data Point Alias dialog, enter the alias name and the formula.
 - i If the data point returns values in the desired units, then leave the value for formula blank.

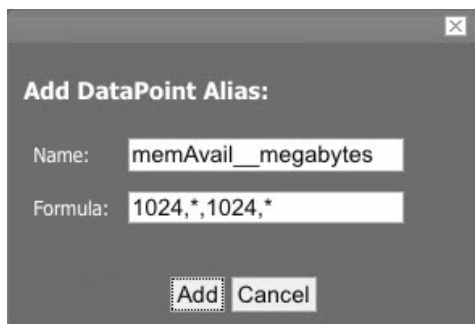


Figure 6.18. Data Point Alias Dialog

5. Click **Add** to add the data point alias.

6.2.6.3. Zenoss Reports That Use Aliases

For information about Zenoss reports that use aliases, refer to the chapter titled "Reporting."

The following table shows Zenoss performance reports that use aliases, and the aliases used. To add data points to a report, add the alias, and then ensure the values return in the expected units.

CPU Utilization

| Alias | Expected Units |
|-----------------|----------------|
| loadAverage5min | Processes |

| Alias | Expected Units |
|---------|----------------|
| cpu_pct | Percent |

6.2.7. Thresholds

Thresholds define expected bounds for data points. When the value returned by a data point violates a threshold, Zenoss creates an event.

Min/Max Threshold

Zenoss provides one built-in threshold type: the Min/Max Threshold. (Other threshold types are provided through ZenPacks.)

Min/Max thresholds inspect incoming data to determine whether it exceeds a given maximum or falls below a given minimum. You can use a Min/Max threshold to check for these scenarios:

- *The current value is less than a minimum value.* To do this, you should set only a minimum value for the threshold. Any value less than this number results in creation of a threshold event.
- *The current value is greater than a maximum value.* To do this, you should set only a maximum value for the threshold. Any value greater than this number results in creation of a threshold event.
- *The current value is not a single, pre-defined number.* To do this, you should set the minimum and maximum values for the threshold to the same value. This will be the only "good" number. If the returned value is not this number, then a threshold event is created.
- *The current value falls outside a pre-defined range.* To do this, you should set the minimum value to the lowest value within the good range, and the maximum value to the highest value within the good range. If the returned value is less than the minimum, or greater than the maximum, then a threshold event is created.
- *The current value falls within a pre-defined range.* To do this, you should set the minimum value to the highest value within the bad range, and the maximum value to the lowest value within the bad range. If the returned value is greater than the maximum, and less than the minimum, then a threshold event is created.

Adding Thresholds

Follow these steps to define a Min/Max threshold for a data point.

1. From the Performance Template page, select Add a Threshold from the Thresholds table menu.

The Add a New Threshold dialog appears.

2. Enter an ID for the new threshold, and then click **OK**.

The Min/Max Threshold page appears.

Min/Max Threshold

State at time: 2008/06/18 13:47:35

| | |
|----------------|--|
| Name | CPU Utilization |
| Data Points | laLoadInt5_laLoadInt5 memAvailReal_memAvailReal memAvailSwap_memAvailSwap memBuffer_memBuffer memCached_memCached ssCpuRawIdle_ssCpuRawIdle ssCpuRawSystem_ssCpuRawSystem ssCpuRawUser_ssCpuRawUser ssCpuRawWait_ssCpuRawWait sysUpTime_sysUpTime |
| Min Value | 2 |
| Max Value | |
| Event Class | /Perf/CPU |
| Severity | Warning |
| Escalate Count | 5 |
| Enabled | True |
| Save | |

Figure 6.19. Add a Threshold

3. Enter or select values to define the threshold:

- **Name** - Displays the value for the ID you entered on the Add a New Threshold dialog. This name appears on the Performance Template page.
- **Data Points** - Select one or more data points to which this threshold will apply.
- **Min Value** - If this field contains a value, then each time one of the select data points falls below this value an event is triggered. This field may contain a number or a Python expression.

When using a Python expression, the variable `here` references the device or component for which data is being collected. For example, an 85% threshold on an interface might be specified as:

```
here.speed * .85/8
```

The division by 8 is because interface speed frequently is reported in bits/second, where the performance data is bytes/second.

- **Max Value** - If this field contains a value, then each time one of the selected data points goes above this value an event is triggered. This field may contain a number or a Python expression.
- **Event Class** - Select the event class of the event that will be triggered when this threshold is breached.
- **Severity** - Select the severity level of the first event triggered when this threshold is breached.
- **Escalate Count** - Enter the number of consecutive times this threshold can be broken before the event severity is escalated by one step.
- **Enabled** - Select True to enable the threshold, or False to disable it.

4. Click **Save** to save the threshold.

6.2.8. Performance Graphs

You can include any of the data points or thresholds from a performance template in a *performance graph*.

To define a graph:

1. Navigate to a performance template whose data you want represented in a graph.

- From the Graph Definitions table menu, select Add Graph.

The Add a New Graph dialog appears.

- Enter the name of the graph, and then click **OK**.

The Graph Definition page appears.

- Enter information or select values to define the graph:

- **Name** - Optionally edit the name of the graph you entered in the Add a New Graph dialog. This name appears as the title of the graph.
- **Height** - Enter the height of the graph, in pixels.
- **Width** - Enter the width of the graph, in pixels.
- **Units** - Enter a label for the graph's vertical axis.
- **Logarithmic Scale** - Select True to specify that the scale of the vertical axis is logarithmic. Select False (the default) to set the scale to linear. You might want to set the value to True, for example, if the data being graphed grows exponentially. Only positive data can be graphed logarithmically.
- **Base 1024** - Select True if the data you are graphing is measured in multiples of 1024. By default, this value is False.
- **Min Y** - Enter the bottom value for the graph's vertical axis.
- **Max Y** - Enter the top value for the graph's vertical axis.
- **Has Summary** - Select True to display a summary of the data's current, average, and maximum values at the bottom of the graph.

Graph Definition

Graph Custom Definition Graph Commands

Graph Points

| Seq | Name | Type | Description |
|-----|-------------------------------------|-----------|-----------------------|
| 0 | <input type="checkbox"/> laLoadInt5 | DataPoint | laLoadInt5_laLoadInt5 |

State at time: 2008/06/18 11:01:23

| | |
|-------------------|--------------------|
| Name | Load Average 5 min |
| Height | 100 |
| Width | 500 |
| Units | processes |
| Logarithmic Scale | False |
| Base 1024 | False |
| Min Y | -1 |
| Max Y | -1 |
| Has Summary | True |

Save

Figure 6.20. Graph Definition

6.2.8.1. Graph Points

Graph points represent each data point or threshold that is part of a graph. You can add any number of graph points to a graph definition by adding data points or thresholds.

From the Graph Points table menu:

- Select Add DataPoint, Add Threshold, or Add Custom.

2. Select values for the graph point, and then click **OK**.

The new graph point appears on the Graph Definition page.

-  Thresholds are always drawn before other graph points.

6.2.8.1.1. Re-sequencing Graph Points

To re-sequence graph points, enter a sequence number in one or more Seq fields and then select Re-sequence GraphPoints from the the Graph Points table menu. The graphs points are re-ordered as specified.

6.2.8.1.2. DataPoint Graph Points

DataPoint graph points draw the value of data points from the template on a graph.

6.2.8.1.2.1. Adding DataPoint Graph Points

To define a DataPoint graph point:

1. From the Graph Points table menu, select Add DataPoint.

The GraphPoint dialog appears.

2. Select one or more data points defined in this template. One DataPoint graph point is created for each data point you select from the list.
3. Optionally, select the Include Related Thresholds option. If selected, then any graph points are created for any thresholds that have been applied to the select data points as well.
4. Click **OK** to add the graph point.

6.2.8.1.2.2. Editing DataPoint Graph Points

Click the name of the graph point to go to its edit page. Enter information or select values to edit the graph point:

- **Name** - This is the name that appears on the Graph Definition page. By default, it appears in the graph legend.
- **Consolidation** - Specify the RRD function used to graph the data point's data to the size of the graph. Most of the time, the default value of AVERAGE is appropriate.
- **RPN** - Optionally enter an RPN expression that alters the value of the data being graphed for the data point. For example, if the data is stored as bits, but you want to graph it as bytes, enter an RPN value of "8,/" to divide by 8. For more information about RRDTool RPN notation, go to:

<http://oss.oetiker.ch/rrdtool/tut/rpntutorial.en.html>

- **Limit** - Optionally specify a maximum value for the data being graphed.
- **Line Type** - Select Line to graph the data as a line. Select Area to fill the area between the line and the horizontal axis with the line color. Select None to use this data point for custom RRD commands and do not want it to be explicitly drawn.
- **Line Width** - Enter the pixel width of the line.
- **Stacked** - If True, then the line or area is drawn above the previously drawn data. At any point in time on the graph, the value plotted for this data is the sum of the previously drawn data and the value of this data point now. You might set this value, for example, to asses total packets if measuring packets in and packets out.
- **Color** - Optionally specify a color for the line or area. Enter a six-digit hexadecimal color value with an optional two-digit hex value to specify an alpha channel. An alpha channel value is only used if 'stacked' is True.
- **Format** - Specify the RRD format to use when displaying values in the graph summary. For more information on RRDTool formatting strings, go to:

http://oss.oetiker.ch/rrdtool/doc/rrdgraph_graph.en.html

- **Legend** - Name to use for the data in the graph legend. By default, this is a TALEX expression that specifies the graph point name. The variables available in this TALEX expression are here (the device or component being graphed) and graphPoint (the graph point itself).
- **Available RRD Variables** - Lists the RRD variables defined in this graph definition. These values can be used in the RPN field.

6.2.8.1.2.3. Editing Threshold Graph Points

Threshold graph points graph the value of thresholds from the template.

6.2.8.1.3. Threshold Graph Points

Threshold graph points graph the value of thresholds from the template.

To add a threshold graph point to the graph definition:

1. Select Add Threshold from the Graph Points table menu.

The Add GraphPoint dialog.

2. Select one or more thresholds defined in this template. One threshold graph point is created for each threshold you select in this list.

You can edit values for Name, Color, and Legend for a threshold graph point. Refer to the definitions in the section titled Editing DataPoint Graph Points for more information.

6.2.8.1.4. Custom Graph Points

Custom graph points allow you to insert specific RRD graph commands into the graph definition.

For details on DEF, CDEF, and VDEF commands, go to:

http://oss.oetiker.ch/rrdtool/doc/rrdgraph_data.en.html

For details on other RRD commands, go to:

http://oss.oetiker.ch/rrdtool/doc/rrdgraph_graph.en.html

6.2.8.2. Custom Graph Definition

The Custom Graph Definition tab allows you to specify your own set of RRD commands to draw a graph. The graph points specified on the Custom Graph Definition tab are used to define data that is available to the commands you specify here; however, the graph points are not drawn unless you explicitly draw them with the commands you specify here. The Available RRD Variables lists the values defined by the graph points that are available for use.

6.2.8.3. Graph Commands

The Graph Commands tab shows an approximate representation of the RRD commands that will be used to draw a graph. This representation provides helpful debugging information when using custom graph points or the Custom Definition tab.

6.2.9. Changing Graph Display Order

You can change the sequence of the appearance of graphs. To do this:

1. Navigate to a device.
2. From the page menu, select More > Templates.
3. Click **Create Local Copy**.
4. Click the name of the template.

5. In the Graphs area of the page, use the Seq options to order the graphs.

6.3. Monitoring Using ZenCommand

Read the following sections for more information about monitoring using ZenCommand.

6.3.1. About ZenCommands

Zenoss has the ability to run Nagios® and Cacti plug-ins through the ZenCommand process. ZenCommand can run plugins locally and remotely by using a native SSH transport. When run, Zenoss tracks the return code of each plug-in and then creates events with plug-in output. Additionally, Zenoss can track performance information from a plug-in.

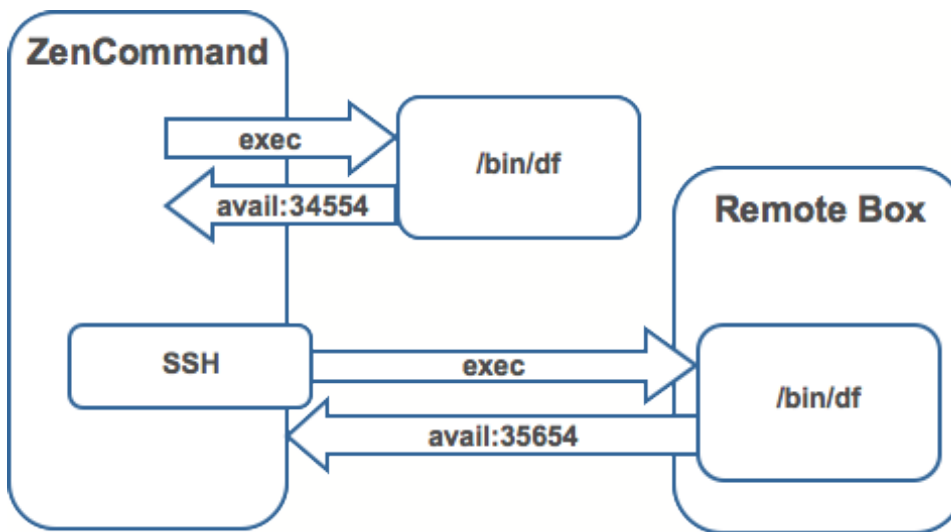


Figure 6.21. Running ZenCommands

6.3.2. Example: Writing a ZenCommand (check_http example)

You can use a ZenCommand plugin (check_http) to check for specific content of a Web page. (This implicitly checks for server/page 200 status as well.)

The following example procedure shows how to set up a ZenCommand plugin to check specific content. The steps show how to test the plugin, and then integrate it with Zenoss.

1. As the zenoss user, test the plugin from the command line. Enter the following command to test the product directory:

```
$ZENHOME/libexec/check_http -H www.zenoss.com
```

If the check_http command is correct, the output will look similar to the following.

```
HTTP OK HTTP/1.0 200 OK - 0.723 second response time |time=0.722758s;;;0.000000 size=7932B;;;0
```

i The `check_http -h` command displays all plugin options.

2. Add the device you want to check (one running a "www" Web site) to the Zenoss system, setting the discovery protocol to "none."
3. Navigate to the device in Zenoss, and then select More > Templates from the page menu.
4. Click **Create Local Copy**.

The default device template is overridden with the device template specific to this device.

5. In the template, remove the sysUpTime data source. (In this example, SNMP is not used for the device.)

6. Add a new description to the template.
7. Add a new data source called rootWebCheck.
8. In the rootWebCheck data source, set the following variables:

- Source Type = COMMAND
- Component = rootWebCheck
- Cycle Time = 30

9. Debug your ZenCommand by running zencommand in the foreground with debugging on:

```
zencommand run -d www.website.com -v10
```

Where *www.website.com* is the site you want to monitor.

The command template field is a TALEX expression. You can make substitutions in the command that will make it generic for any device added to this class.

10. Set the `-H` flag to the IP of the device against which this command will be run, as follows:

```
check_http -H ${here/manageIp}
```

11. Add a check looking for content on the page. The `-r` flag will run a regular expression against the Web page to check for text.

```
check_http -H ${here/manageIp} -r textstring1
```

Where *textstring1* is text you know is on the resulting Web page.

12. For this example, the command should be generic. Make a custom field for the regex that can be changed per device. Set the default to `".*"`, which will match everything. Go to /Devices/Custom Schema and add a new field:

- Label = Web Match Regex
- Name = cWebMatchRegex
- Type = string
- Default = `.*`
- Visible = True

13. Return to the template and change the command to be.

```
check_http -H ${here/manageIp} -r ${here/cWebMatchRegex}
```

14. Add the regex value into cWebMatchRegex (used in the example above).
15. Test the ZenCommand from the command line.

6.3.3. Example: Collect Data from A ZenCommand

To collect and display data from the ZenCommand `check_http` example, you can log the data to see something like response time in a graphical format.

1. Navigate to /Web/Device template.
2. Go to the data source created in the `check_http` example.
3. In the Data Points table, add a data point named "time." (No modifications are needed to the data point.)
4. Test the command again. You should see a log message that starts with:

```
DEBUG:zen.zencommand:storing responseTime = 1.0
```


5. Make a graph to display the data. In the device template, create a graph called "Web Response Time."
6. For this graph, set the following values:
 - Data Sources = rootWebCheck_responseTime

- Units = Seconds
 - Min Y = 0
7. View the Perf tab for the device `www.website.com` (the Web site you were using to check) to see the graph. Graph data will not appear until collection is run several times. Restart `zencommand` so that the new configuration takes effect immediately.

6.3.4. Plugin Format for ZenCommands

Nagios® plugins are configured by using a command template that is much like the RRDTemplates used for performance monitoring.

A template named “Device” will bind to all devices below the template definition. Within each template is a list of commands that will run. The commands can be any program that follows the Nagios® plug-in standard. Inputs are command line arguments; output is the first line of stdout, plus a return code.

 Zenoss return codes differ from Nagios return codes, as follows:

| Value | Zenoss | Nagios |
|-------|---------------|----------|
| 0 | Clear | OK |
| 1 | Data Source | WARNING |
| 2 | Data Source+1 | CRITICAL |
| 3 | Data Source | UNKNOWN |

For complete information about Nagios plugin guidelines, browse to this location:

<http://nagiosplug.sourceforge.net/developer-guidelines.html>

A Nagios® command has several fields:

- name – Specifies the name of the command object.
- enabled – Indicates whether this command should be used on a given device.
- component – Specifies the component name to use when `zencommand` sends events to Zenoss.
- event class – Specifies the event class to use when sending events to Zenoss.
- severity – Sets the default severity to use when sending events to Zenoss.
- cycle time – Sets the frequency a command should be run (in seconds).
- command template – Specifies the command to run.

The command template string is built by using Zope TALEs expressions. Several variables are passed when evaluating the template. They are:

- `zCommandPath` – Path to the `zencommand` plug-ins on a given box it comes from the `zProperty zCommandPath`. `zCommandPath` is automatically added to a command if a path is absent from the beginning of the command.
- `devname` – Device name of the device against which the command is being evaluated.
- `dev` – Device object against which the command is being evaluated.
- `here` – Context of evaluation. For a device, this is equivalent to `dev` for a component (such as a filesystem or interface). This is the component object.
- `compname` – If this command evaluates against a component, specifies its name as a string.
- `now` – Current time.

Template values are accessed like shell variables. They are the same as the expression syntax used in the appendix titled TALEs Expressions (in this guide).

Examples

Run an http check against all devices by using the URL /zport/dmd:

```
check_http -H ${devname} -u /zport/dmd
```

In a template named FileSystem, the following command will run against all file systems on a device:

```
check_disk -w 10% -c 5% -p ${compname}
```

6.3.5. Testing ZenCommands

You can test ZenCommand data sources by using the zentestcommand shell script.

From the command line, run:

```
zentestcommand -d DeviceName --datasource=DataSourceName
```

where *DeviceName* is the device on which you want to run the command, and *DataSourceName* is the name of a data source on a template associated with the device.

The zentestcommand script prints the results of the command to standard output.

6.4. SNMP Monitoring

OID represent the data points where the data for the graphs comes from. Sometimes the reason that a graph is not appearing is because the OID for the particular graph is not valid for the device. You can test this validity using the command line to see if you can return a value. To test the validity of an OID data point giving performance data:

1. SSH to the Zenoss instance.

Use Username: root

Password: zenoss

2. Run the command snmp get for one of the OIDs

In this case, use the command:

```
$ snmpget -v1 -cpublic build .1.3.6.1.4.1.2021.4.14.0
```

If the OID is valid it will return a value.

Here are some basic SNMP commands to gather certain information.

- a. Walk a basic system MIB.

```
snmpwalk -v1 -cpublic <device name> system
```

- b. Walk an interface description

```
snmpwalk -v1 -cpublic <device name> ifDescr
```

- c. Get a single value.

```
snmpget -v1 -cpublic <device name> ifDescr.2
```

- d. Detailed description of an OID value.

```
snmptranslate -Td RFC1213-MIB::ifDescr
```

- e. Convert a name to a raw OID.

```
snmptranslate -On RFC1213-MIB::ifDescr
```

- f. Convert a raw OID to a short name

```
snmptranslate -OS .1.3.6.1.2.1.2.2.1.2
```

6.5. Monitoring Devices Remotely Through SSH

You can monitor devices remotely through SSH. To monitor devices remotely, you must install the Zenoss plugins on each remote device you want to monitor.

Follow the steps in the following sections to set up remote monitoring.

6.5.1. Installing Zenoss Plugins on the Remote Machine

The Zenoss Plugins are packaged in two formats:

- **Native format (RPM)** - Recommended in Red Hat-based systems that support Red Hat Package Management. By using the RPM distribution, you can easily update the package when newer versions are released.
- **Source distribution** - Assembled using `setuptools`. When using the source distribution, you do not need root privileges to install the Zenoss plugins.

6.5.1.1. Zenoss Plugin Installation Technique: RPM

The RPM for the Zenoss Plugins is a noarch RPM, which means it can be installed on any architecture (such as i386, amd64, or ia_64). The only external dependency needed to install the Zenoss plugins RPM is Python. Most Linux distributions include Python in their standard loads.

To install the Zenoss plugins RPM, use the following command:

```
$ sudo rpm -Uvh zenoss-plugins-*.rpm
```

where 'zenoss-plugins-*.rpm' is the latest Zenoss plugin RPM file.

6.5.1.2. Zenoss Plugin Installation Technique: setuptools

Enter these commands to install the Zenoss plugins into directories that are accessible to all users:

```
$ python setup.py build
```

```
$ sudo python setup.py install
```

If you do not have appropriate privileges to install the system software, refer to the following information about installing the plugins using a non-privileged account, at:

<http://dev.zenoss.org/trac/wiki/ZenossPlugins>

Alternatively, you can use `setuptools`'s built-in `easy_install` command to install the plugins. To use `easy_install` to download and install the Zenoss plugins, run the following command:

```
$ sudo easy_install Zenoss-Plugins
```

where 'Zenoss-Plugins' is the name of the latest Zenoss plugin file.

6.5.1.3. Testing the Plugin Installation

The entry point to the Zenoss plugins is the `zenplugin.py` command. When run without any arguments, `zenplugin.py` reports the proper usage of the script, providing insight into which options should be run for troubleshooting.

The Zenoss plugins detect platform-specific, runtime values using plugins. For example, the CPU plugin for the linux2 platform uses `/proc` to read values. In comparison, the CPU plugin for the freebsd5 platform uses a different technique. In order to test the installation you must determine which plugins are available for your platform. To do this, run the following command:

```
$ zenplugin.py --list-plugins
```

After determining a list of supported plugins for your platform, run the `zenplugin.py` with the plugin name as the argument. The following command line illustrates:

```
$ zenplugin.py cpu
```

6.5.1.4. Troubleshooting Plugin Installation

6.5.1.4.1. "Command not found" when running zenplugin.py

If you receive a "command not found" error when running the zenplugin.py command, make sure that the directory into which zenplugin.py was installed is included in your PATH. If you installed by using RPM, you can use the command "rpm -ql zenoss-plugins | grep zenplugin.py". If you installed via setuptools pay close attention to the "Installing..." messages to see the full directory paths.

6.5.1.4.2. "platform 'XXX' is not implemented. no plugins exist"

This message indicates that Zenoss plugins may not be fully implemented for your particular platform. If you receive this message, and want to investigate support for your platform, email the output of the following command to the Zenoss team:

```
$ python -c 'import sys; print sys.platform'
```

6.5.1.5. Changing Zenoss to Monitor Devices Remotely Using SSH

You must edit Zenoss properties for the group where you want to collect remote information using SSH.

1. Navigate to the device class path you want to monitor remotely. You can apply this monitoring per device or per device class path.
2. Change the zProperties value for the group. Click the zProperties tab.

The zProperties tab appears.

| Property | Value | Type | Path |
|---------------------------|----------------------|---------|------|
| zAxlPassword | ***** | string | / |
| zAxlUsername | administrator | string | / |
| zCollectorClientTimeout | 180 | int | / |
| zCollectorDecoding | latin-1 | string | / |
| zCollectorLogChanges | True | boolean | / |
| zCollectorPlugins | Edit | lines | / |
| zCommandCommandTimeout | 15.0 | float | / |
| zCommandCycleTime | 60 | int | / |
| zCommandExistenceTest | test -f %s | string | / |
| zCommandLoginTimeout | 10.0 | float | / |
| zCommandLoginTries | 1 | int | / |
| zCommandPassword | | string | / |
| zCommandPath | /opt/zenoss/libexec | string | / |
| zCommandPort | 22 | int | / |
| zCommandProtocol | ssh | string | / |
| zCommandSearchPath | | lines | / |
| zCommandUsername | | string | / |
| zDeviceTemplates | Device DnsMonitor | lines | / |
| zFileSystemMapIgnoreNames | | string | / |

Figure 6.22. Device Group zProperties Tab

You must make changes to the following zProperties:

- zCollectorPlugins
- zCommandPassword
- zCommandPath
- zCommandUsername
- zSnmpMonitorIgnore
- zTransportPreference

The following table lists sample values set up for remote devices. These have a pre-shared key (with no password) setup from the collector to the remote boxes. (It can also use password authorization if the password is entered into zCommandPassword.)

| zProperties | Value |
|----------------------|--|
| zCollectorPlugins | snmp portscan |
| zCommandPassword | The SSH password for the remote machine. |
| zCommandPath | The path to zenplugin.py |
| zCommandUsername | The SSH Username for the remote machine. |
| zSnmpMonitorIgnore | True |
| zTransportPreference | command |

Two passes are required for full modeling. The first pass obtains the platform type (so that Zenoss knows which plugins to run). The second pass provides detailed data on interfaces and file systems.

Run the command:

```
$ zenmodeler run -d enter_server_name_here
```

Run the command a second time to use the plugins the command gathered on the first pass.

6.5.1.6. Using the Predefined /Server/Cmd Device Class

The /Server/Cmd device class is an example configuration for modeling and monitoring devices using SSH. The zProperties have been modified as described in the previous sections, and Device, Filesystem and Ethernet interface templates that gather data over SSH have been created.

You can use this device class as a reference for your own configuration; or, if you have a device that needs to be modeled or monitored via SSH/Command, you can place it under this device class to use the pre-configured templates and zProperties. You will still need to set the zCommandUsername and zCommandPassword zProperties to the appropriate SSH login information for each device.

6.6. Monitoring Windows Devices

6.6.1. Device Preparation for Windows Devices

In all Zenoss versions, WMI is used to monitor the Windows event log and state of Windows services.

Before you can monitor Windows devices with Zenoss, you must ensure that:


- DCOM is enabled for WMI connections
- The hostname of the Zenoss collector does not exceed fifteen characters

If you are using Zenoss Core, you must additionally ensure that an SNMP agent is enabled on Windows devices. If your system is running Windows Vista, for example, follow these steps to see if the SNMP agent is enabled:


1. From the Start menu list, right-click Computer, and then select Manage from the list of options.
2. From the Computer Management panel navigation area, expand Services and Applications, and then select Services.

The Services list appears.

3. Locate the listing for SNMP Service. If it does not show a status of "Started," then click Start (the service).

 If SNMP Service does not appear in the list, then you may have to enable the SNMP feature (from the "Turn Windows features on and off" selection in the Control Panel).

Optionally, you can use SNMP Informant™ to collect CPU, memory, and disk I/O statistics. SNMP Informant agents collect information from Windows devices via WMI on the server where they are installed, and then convert system, state, and operational data into SNMP OIDs for broadcast. Zenoss can then process the SNMP OID information and generate events and alerts based on this information. See the section titled Monitoring Windows Performance with SNMP Informant (in this chapter) for more information.

 If you are using Zenoss Enterprise, SNMP Informant is not needed (its functionality is included in these versions).

6.6.2. Setting Windows zProperties

You must set the following zProperties to collect information from Windows servers. In Zenoss, navigate to the zProperties for each device, and then set the appropriate values for:

- **zWmiMonitorIgnore** - Turns on or off all WMI monitoring. Set the value of Ignore to False to turn on Windows monitoring.

Zenoss recommends that you set this zProperty at the Server/Windows class level, so that any device placed in this class has Windows monitoring automatically enabled.

- **zWinUser** - Must be set as the local admin. The format for zWinUser is:
 - .\Username - The format to use when the account is a local account.
 - DOMAIN\Username - The format for a Domain account.
- **zWinPassword** - Enter the password used to remotely log in to the Windows machine.

6.6.3. Testing WMI on a Windows Server

Follow these steps to test the WMI connections on the Windows server:

1. Run wbemtest.
2. Click "Connect..."
3. In the Namespace field, enter:

```
\\HOST\root\cimv2
```

4. Enter login information in the User and Password fields.
5. Click **Query**.
6. Enter "select * from win32_service" to return a dialog with a list of services on the device.

6.6.4. Optional Windows Configuration

Zenoss can gather additional, detailed OS and hardware information from Windows devices if you have these agents installed on your Windows device:

- Dell Open Manage Agent
- HP Insight Management Agent


6.6.5. Modeling Services on Windows Devices

Zenoss uses ZenWin to perform Windows Service Monitoring over WMI. ZenWin monitors the up and down availability of Windows services.

The WinServiceMap WMI plugin is included in zCollectorPlugins on the /Server/Windows device class. WinServiceMap retrieves all services that can be monitored on a device, regardless of whether it is up or down.

Windows services are (by default) not monitored. To monitor a specific Windows service, follow these steps:

1. In Zenoss, navigate to the Windows device, and then click the OS tab.
2. Click the service you want to monitor, and then set the value of monitor to True.

 If you do not see the service you want to monitor in the list, then you can add it. Select Add WinService from the WinServices table menu.

6.6.6. Collecting Windows Eventlog Events

Zenoss uses ZenEventlog to collect WMI event log events. Enable the following zProperties to define how Windows event log events are processed and monitored:

- **zWinEventLog** - Tells Zenoss whether or not to read the event log into the system.
- **zWinEventLogMinSeverity** - Sets the minimum severity to collect from the Windows event log. The lowest number indicates the highest severity (1 is the most severe; 5 is least severe).

6.6.7. Monitoring Windows Performance with SNMP Informant

Zenoss can use information from SNMP Informant to collect SNMP information from Windows devices.

Install the free version of SNMP Informant from this location:

<http://www.snmp-informant.com>

To make sure SNMP Informant is running and set up correctly, run this command to walk the SNMP Informant MIB:

```
snmpwalk -v1 -c<community> <server> 1.3.6.1.4.1.9600
```

This command will return some performance information if SNMP Informant is configured and running correctly.

Once this is configured properly, Zenoss gathers and uses SNMP information the same as any other device sending SNMP traps.

6.6.8. Running winexe Commands on Windows Servers

You can use winexe commands to run commands on monitored Windows servers from within Zenoss.

Usage:

```
$ZENHOME/bin/winexe [options] //host [command]
```

| Options | Use |
|-----------------------------------|--|
| --uninstall | Uninstall winexe service after remote execution. |
| --reinstall | Reinstall winexe service before remote execution. |
| --system | Use SYSTEM account. |
| --runas=[DOMAIN]USERNAME%PASSWORD | Run as user (IMPORTANT! password is sent in cleartext over net). |

| Help Options | Use |
|--------------|------------------------------|
| -?, --help | Show this help message. |
| --usage | Display brief usage message. |

| Common samba options | Use |
|--------------------------------|--|
| -d, --debuglevel=DEBUGLEVEL | Set debug level. |
| --debug-stderr | Send debug output to STDERR. |
| -s, --configfile=CONFIGFILE | Use alternative configuration file. |
| --option=name=value | Set smb.conf option from command line. |
| -l, --log-basename=LOGFILEBASE | Basename for log/debug files. |
| --leak-report | enable talloc leak reporting on exit. |
| --leak-report-full | enable full talloc leak reporting on exit. |
| -V, --version | Print version. |

| Connection Options | Use |
|---------------------------------------|--|
| -R, --name-resolve=NAME-RESOLVE-ORDER | Use these name resolution services only. |
| -O, --socket-options=SOCKETOPTIONS | Socket options to use. |
| -n, --netbiosname=NETBIOSNAME | Primary netbios name. |
| -W, --workgroup=WORKGROUP | Set the workgroup name. |
| --realm=REALM | Set the realm name. |
| -i, --scope=SCOPE | Use this Netbios scope. |
| -m, --maxprotocol=MAXPROTOCOL | Set max protocol level. |

| Authentication Options | Use |
|---|--|
| -U, --user=[DOMAIN \\USERNAME[%PASSWORD] | Set the network user name. |
| -N, --no-pass | Do not ask for a password. |
| --password=STRING | Password |
| -A, --authentication-file=FILE | Get the credentials from a file. |
| -S, --signing=on off required | Set the client signing state. |
| -P, --machine-pass | Use stored machine account password (implies -k). |
| --simple-bind-dn=STRING | DN to use for a simple bind. |
| -k, --kerberos=STRING | Use Kerberos. |
| --use-security-mechanisms=STRING | Restricted list of authentication mechanisms available for use with this authentication. |

Chapter 7. Event Management

7.1. About Events

Events, and the graphs generated from performance monitoring, are the primary operational tools for understanding the state of your environment. This chapter defines events and describes the Zenoss event management system.

7.1.1. Basic Event Fields

To enter the Zenoss event management system, an event must contain values for the device, severity, and summary fields. If an event is missing any of these fields, then Zenoss rejects it.

Basic event fields are:

- device
- ipAddress
- eventState
- severity
- summary
- message
- evid

7.1.1.1. device and ipAddress Fields

The device field is a free-form text field that allows up to 128 characters. Zenoss accepts any value for this field, including devices that are not in the database. If the device field contains an IP address, then Zenoss queries the database for devices with a matching address. If it finds a match, it changes the device field to the found device name.

The ipAddress field is a free-form text field that allows up to 15 characters. This field is not required. If Zenoss cannot successfully locate a device based on the event's device field content, it attempts to find the device based the event ipAddress field content, if present.

Zenoss automatically adds information to incoming events that match a device in its database. Fields added are:

- **prodState** - Specifies the device's current production state.
- **Location** - Specifies the location (if any) to which the device is assigned.
- **DeviceClass** - Classifies the device.
- **DeviceGroups** - Specifies the groups (if any) to which the device is assigned.
- **Systems** - Systems (if any) to which the device is assigned.
- **DevicePriority** - Priority assigned to the device.

For more information about these fields, refer to the chapters titled "Production States and Maintenance Windows" and "Organizers and Path Navigation."

7.1.1.2. eventState Field

The eventState field defines the current state of the event. This field is often updated after an event has been created. Values for this numeric field are 0-2, defined as follows:

| Number | Name |
|--------|------|
| 0 | New |

| Number | Name |
|--------|--------------|
| 1 | Acknowledged |
| 2 | Suppressed |

7.1.1.3. severity Field

The severity field defines the severity of the event. Values for this numeric field are 0-5, defined as follows:

| Number | Name | Color |
|--------|----------|--------|
| 0 | Clear | Green |
| 1 | Debug | Grey |
| 2 | Info | Blue |
| 3 | Warning | Yellow |
| 4 | Error | Orange |
| 5 | Critical | Red |

7.1.1.4. summary and message Fields

The summary and message fields are free-form text fields. The summary field allows up to 128 characters. The message field allows up to 65535 characters. These fields usually contain similar data.

Zenoss handles these fields differently, depending on whether one or both are present on an incoming event:

- If only summary is present, then Zenoss copies its contents into message and truncates summary contents to 128 characters.
- If only message is present, then Zenoss copies its contents into summary and truncates summary contents to 128 characters.
- If summary and message are both present, then Zenoss truncates summary contents to 128 characters.

As a result, data loss is possible only if the message or summary content exceeds 65535 characters, or if both fields are present and the summary content exceeds 128 characters.

To ensure that enough detail can be contained within the 128-character summary field limit, avoid reproducing information in the summary that exists on other fields (such as device, component, or severity).

7.1.1.5. evid

The evid field is the event identifier, or event ID. It is a 36-character, unique identifier for every event that comes into the system. An incoming event should never have an evid assigned to it, because Zenoss creates it immediately before the event is inserted into the database. If an incoming event does have an assigned evid, then Zenoss ignores it and replaces it with a generated evid.

7.1.2. Other Fields

Zenoss events include numerous other standard fields. Some control how an event is mapped and correlated; others provide information about the event.

The following table lists additional event fields.

| Field | Description |
|-----------|--|
| depuuid | Dynamically generated fingerprint that allows Zenoss to perform de-duplication on repeating events that share similar characteristics. |
| component | Free-form text field (maximum 255 characters) that allows additional context to be given to events (for example, the interface name for an interface threshold event). |

| Field | Description |
|-------------------|---|
| eventClass | Name of the event class into which this event has been created or mapped. |
| eventKey | Free-form text field (maximum 128 characters) that allows another specificity key to be used to drive the de-duplication and auto-clearing correlation process. |
| eventClassKey | Free-form text field (maximum 128 characters) that is used as the first step in mapping an unknown event into an event class. |
| eventGroup | Free-form text field (maximum 64 characters) that can be used to group similar types of events. This is primarily an extension point for customization. Currently not used in a standard Zenoss system. |
| stateChange | Last time that any information about the event changed. |
| firstTime | First time that the event occurred. |
| lastTime | Most recent time that the event occurred. |
| count | Number of occurrences of the event between the firstTime and lastTime. |
| prodState | Production state of the device when the event occurred. If an event is still active when a device's production state is changed, the event's prodState will be updated accordingly. |
| suppid | If this event has been suppressed by another event, then suppid contains the other event's evid. |
| manager | Deprecated. The monitor field replaces this field. |
| agent | Typically the name of the daemon that generated the event. For example, an SNMP threshold event will have zenperfsnmp as its agent. |
| DeviceClass | Device class of the device that the event is related to. |
| Location | Location of the device that the event is related to. |
| Systems | Pipe-delimited list of systems that the device is contained within. |
| DeviceGroups | Pipe-delimited list of systems that the device is contained within. |
| facility | Only present on events coming from syslog. The syslog facility. |
| priority | Only present on events coming from syslog. The syslog priority. |
| ntheid | Only present on events coming from Windows event log. The NT Event ID. |
| ownerid | Name of the user who acknowledged this event. |
| clearid | Only present on events in history that were auto-cleared. The evid of the event that cleared this one. |
| DevicePriority | Priority of the device that the event is related to. |
| eventClassMapping | If this event was matched by one of the configured event class mappings, contains the name of that mapping rule. |
| monitor | In a distributed setup, contains the name of the collector from which the event originated. |

7.1.3. Details

In addition to the standard fields, Zenoss also allows events to add an arbitrary number of additional name/value pairs to events to give them more context. The name and value of these details are limited to 255 characters in length.

7.1.4. De-Duplication

Zenoss uses an event "de-duplication" feature, based on the concept of an event's fingerprint. Within Zenoss, this fingerprint is the "depuid." All of the standard events that Zenoss creates as a result of its polling activities are de-duplicated, with no setup required. However, you can apply de-duplicating to events that arrive from other sources, such as syslog, SNMP traps, or a Windows event log.

The most important de-duplication concept is the *fingerprint*. In all cases, an event's fingerprint (or dedupid) is composed of a pipe-delimited string that contains these event fields:

- device
- component (can be blank)
- eventClass
- eventKey (can be blank)
- severity
- summary (omitted from the dedupid if eventKey is non-blank)

When the component and eventKey fields are blank, a dedupid appears similar to:

www.example.com||/Status/Web||4|WebTx check failed

When the component and eventKey fields are present, a dedupid appears similar to:

router1.example.com|FastEthernet0/1|/Perf/Interface|threshName

When a new event comes into the system, the dedupid is constructed. If it matches the dedupid for any active event, the existing event's count field is incremented by one, and its lastTime field is updated to be the current time. If it does not match the dedupid of any active events, then it is inserted into the active event table with a count of 1, and the firstTime and lastTime fields are set to the current time.

The following illustration depicts a de-duplication scenario in which an identical event occurs three times, followed by one that is different in a single aspect of the dedupid fingerprint.

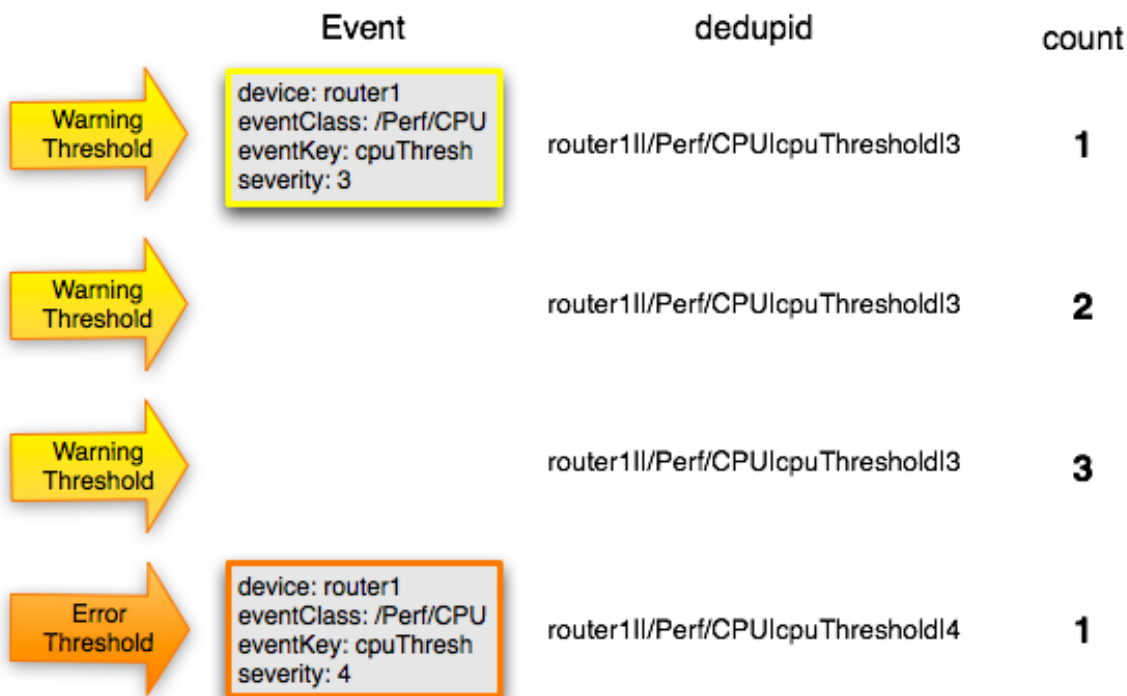


Figure 7.1. Event De-Duplication

If you want to change the way de-duplication behaves in Zenoss, you can use an event transform to alter one of the fields used to build the dedupid. You also can use a transform to directly modify the dedupid field, for more powerful cross-device event de-duplication.

7.1.5. Auto-Clear Correlation

Zenoss' auto-clearing feature is similar to its de-duplication feature. It also is based on the event's fingerprint. The difference is which event fields make up the fingerprint, and what happens when a new event matches an existing event's fingerprint.

All of the standard events that Zenoss creates as a result of its polling activities do auto-clearing by themselves. As with de-duplication, you would invoke auto-clearing manually only to handle events that come from other sources, such as syslog, a Windows event log, or SNMP traps.

The auto-clear fingerprint for an event is built by using the combination of these fields:

- device
- component (can be blank)
- eventKey (can be blank)
- eventClass (including zEventClearClasses from event class zProperties)

When a new event comes into Zenoss with a special 0 (Clear) severity, Zenoss checks all active events to see if they match the auto-clear fingerprint of the new event. All active events that match the auto-clear fingerprint are moved from the active events table to history, and their clearid field is set to the evid of the event that cleared them.

If an event is cleared by the clear event, it is also inserted into the event history; otherwise, it is dropped. This is done to prevent extraneous clear messages from filling your events database.

The following illustration depicts a standard ping down event and its associated clear event.

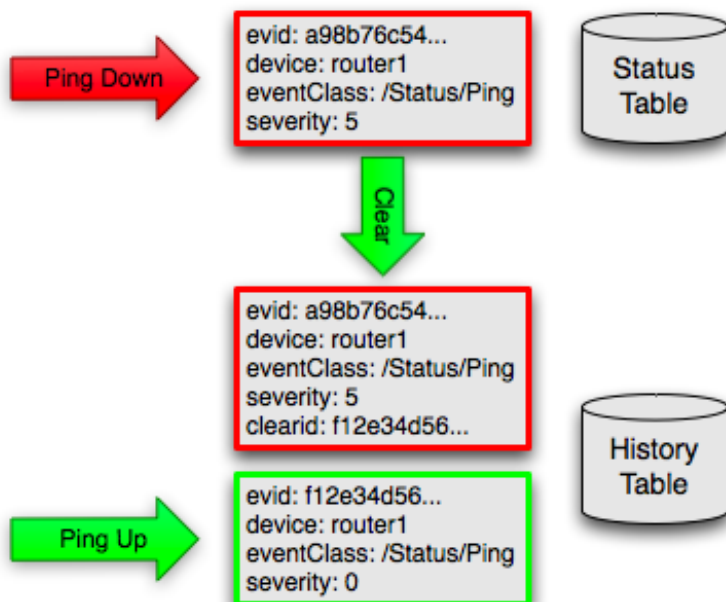


Figure 7.2. Event Auto-Clear

If you need to manually invoke the auto-clearing correlation system, you can use an event transform to make sure that the clear event has the 0 (Clear) severity set. You also need to ensure that the device, component, and eventClass fields match the events you intend to clear.

i Avoid making clear events too generic; otherwise, you may inadvertently clear a wider variety of events that you intend.

7.1.6. Event Consoles

Zenoss features multiple event consoles that allow you to view and manage events. Each console shows different events subsets, depending on your current context.

Zenoss event consoles are:

- **Master** - To access this console, click Event Console in the Navigation menu. You can view all events from this console.

- **Custom** - Users can create custom event consoles from the Event Views tab within their user preferences. Each custom event console has access to the same events as the global console, but can be filtered more specifically (from the Edit tab).
- **Contextual** - Contextual event consoles are found throughout Zenoss. Each time you see an Events tab on a device, device organizer, component, or event class, you can view event information that has been automatically filtered to show events specific to the current context.

The master event console is Zenoss' central nervous system, enabling you to view and manage events. It displays the repository of all events that have been collected by the system.

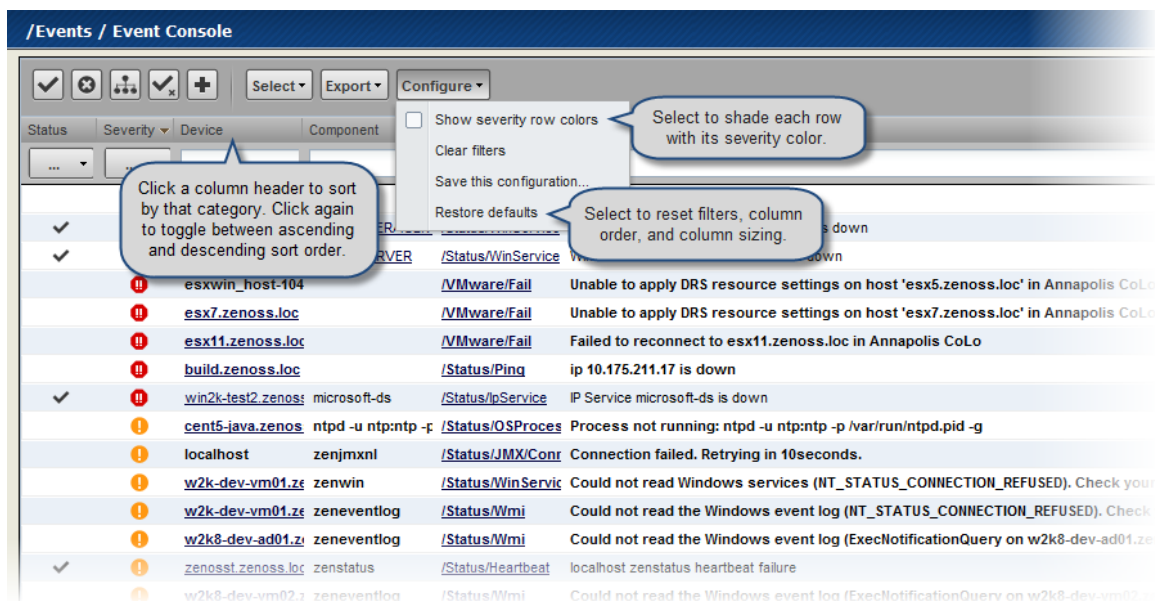


Figure 7.3. Event Console

7.1.6.1. Sorting and Filtering Events

You can sort and filter events that appear in the event console to customize your view.

You can sort events by any column that appears in the event console. To sort events, click a column header. Clicking the header toggles between ascending and descending sort order.

Filter options appear below each column header.

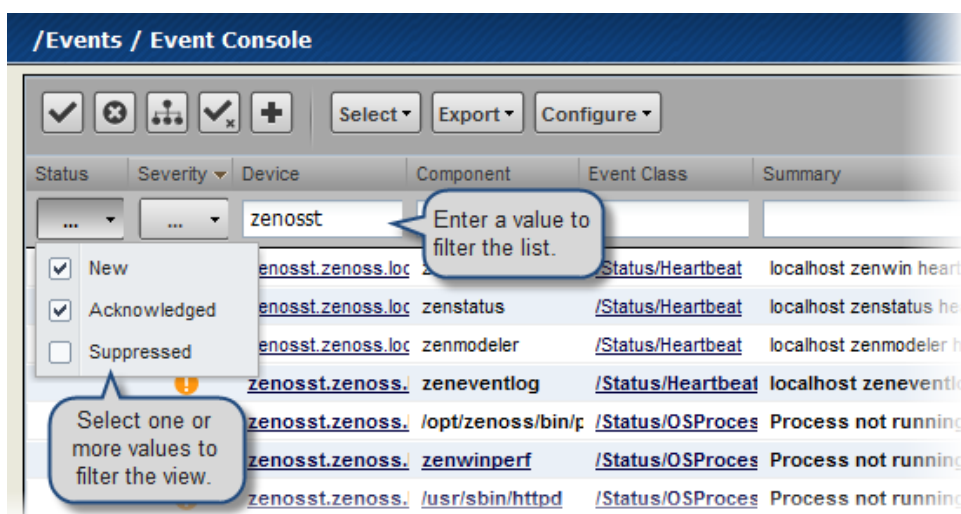


Figure 7.4. Event Console Filter Options

You can filter the events that appear in the list in several ways, depending on the field type. Date fields (such as First Seen and Last Seen) allow you to enter a value or use a date selection tool to limit the list. For other fields, such as Device, Component, and Event Class, enter a match value to limit the list.

The Count field allows you to filter the list when compared to a value:

- n - Displays events with counts greater than or equal to that value.
- $<n$ - Displays events with counts less than that value.
- $\leq n$ - Displays events with counts less than or equal to that value.
- $=n$ - Displays events with counts equal to that value.

To clear filters, select **Configure > Clear filters**.

7.1.6.2. Saving a Custom View

You can save your custom event console view by bookmarking it for quick access later. To do this:

1. Select **Configure > Save this configuration**.

A dialog containing a link to the current view appears.

2. Click and drag the link to the bookmarks link on your browser's menu bar.

Zenoss adds a link titled "Event Console" to your bookmarks list.

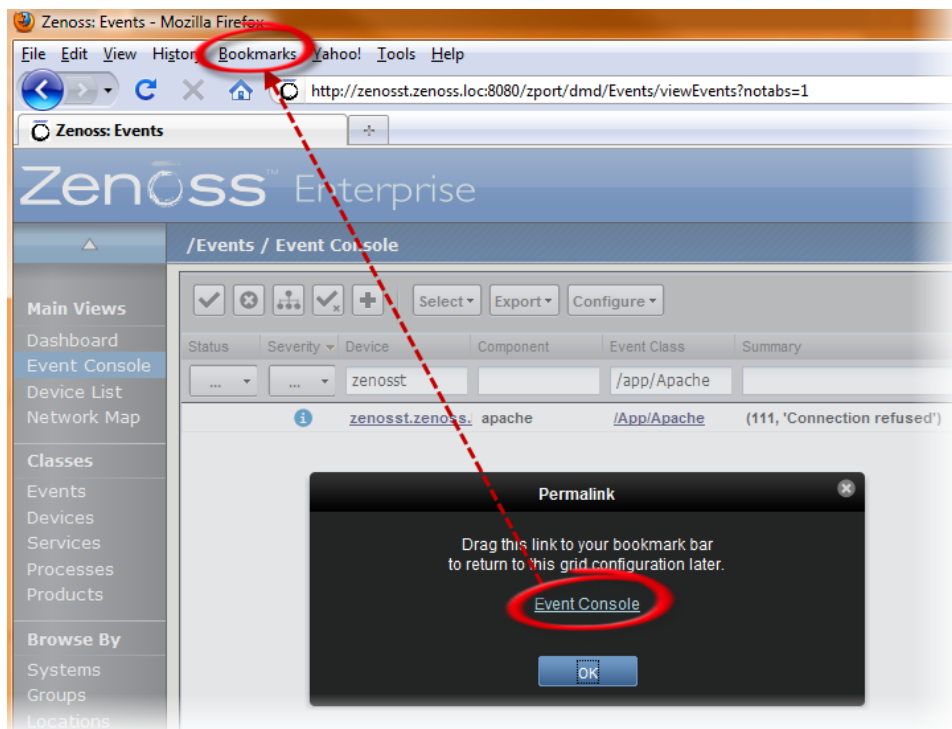


Figure 7.5. Saving a Custom View (Bookmark)

Tip: You may want to re-title the bookmark, particularly if you choose to save more than one event console view.

7.1.6.3. Refreshing the View

You can refresh the list of events manually or specify that they refresh automatically. To manually refresh the view, click **Refresh**. You can manually refresh at any time, even if you have an automatic refresh increment specified.

To set up automatic refresh, select one of the time increments from the Refresh list.

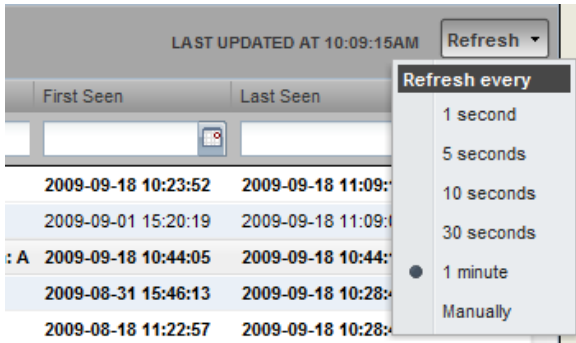


Figure 7.6. Automatic Refresh Selections

7.1.6.4. Viewing Event Details

You can view details for any event in the system. To view details, double-click an event row.

Tip: Be sure not to click other links in the row. These go to other pages.

The Event Details area appears.

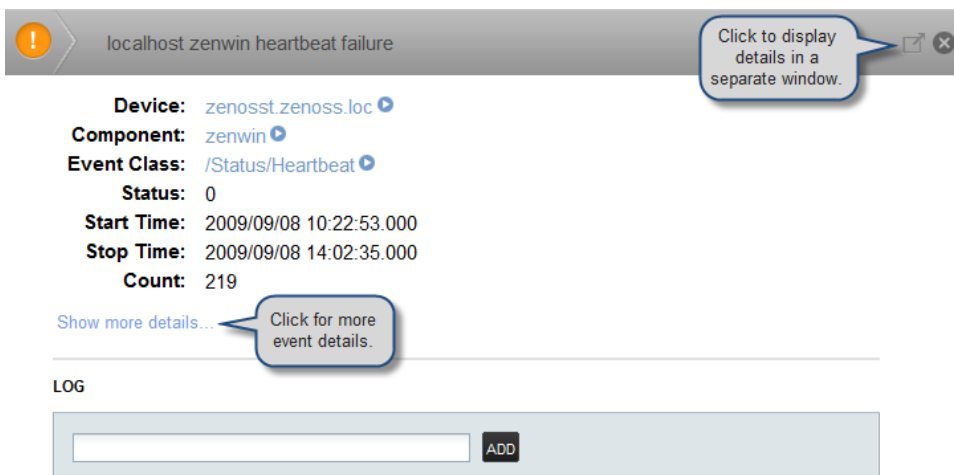


Figure 7.7. Event Details

To see more information about the event, click Show more details. To display the event information in a new window, click the icon located at the top right.

You can use the Log area to add specific information about the event. Enter details, and then click **Add**.


7.1.6.5. Selecting Events

To select one or more events in the list, you can:

- Click a row to select a single event
- Ctrl-Click rows to select multiple events, or Shift-Click to select a range of events
- Click Select to select all, none, new, acknowledged, or suppressed events

7.1.6.6. Acknowledging Events


You may want to mark an event as "acknowledged" to indicate, for example, that you have taken action to remedy a problem. To mark events as acknowledged:

1. Select one or more events in the event console view.
2. Click .

A check mark appears for each acknowledged event.

7.1.6.7. Returning Events to New Status


You may want to return a previously acknowledged event to "new" status (revoke its "acknowledged" status). To do this:

1. Select one or more events in the event console view.
2. Click .

A check mark no longer appears in the event row, and the event is returned to "new" status.


7.1.6.8. Classifying Events

Classifying events lets you associate one or more events with a specific event class. To classify events:

1. Select one or more events in the event console view.
2. Click .

The Classify Events dialog appears.

3. Select an event class from the list of options, and then click **Submit**.


 You can also classify events from event history.

7.1.6.9. Exporting Event Data

You can export data from the event console to a comma-separated value (.csv) or XML file. To do this, select **Export > CSV** or **Export > XML**. By default, the exported file is named `events.Extension`.

7.1.6.10. Moving Events to History (Close)

When you no longer want to actively monitor event (such as after you acknowledge it, for example), you can move it to history. To do this:

1. Select one or more events in the event console view.
2. Click .


The selected events are moved to history.

To view events in history, click the Event History link (located at the bottom left of the Event Console page).

7.1.6.11. Returning Events to Active Status


You can return events that have been moved to history to active status. When you do this, the events reappear in the event console.

To return events in history to active status:

1. Click Event History to go to the event history page.
2. Select one or more events.
3. Click .

The selected events are returned to active status and appear in the event console.

7.1.6.12. Creating Events

To create events from the event console, click .

For more information about manual event creation, see the section titled "Creating Events Manually."

7.1.7. Event Sources

Events come into Zenoss in two ways. *Generated events* are created as a result of active polling. *Captured events* are transmitted by external actions into Zenoss.

7.1.7.1. Generated Events

These standard daemons are responsible for generating events in Zenoss. They automatically perform appropriate de-duplication and auto-clearing.

- **zenping** - Ping up/down events
- **zenstatus** - TCP port up/down events
- **zenperfsnmp** - SNMP agent up/down events, threshold events
- **zencommand** - Generic status events, threshold events
- **zenprocess** - Process up/down events, threshold events
- **zenwin** - Windows service up/down events

7.1.7.2. Captured Events

Captured events are those events that Zenoss does not specifically know will occur in advance. De-duplication is performed on these events, but in some cases may need to be tuned. By default, no auto-clearing is done on captured events. Event transforms must be used to create the auto-clear correlations.

These standard daemons are responsible for collecting captured events:

- **zensyslog** - Events created from syslog messages.
- **zentrap** - Events created from SNMP traps and informs.
- **zeneventlog** - Events created from the Windows event log.

There are a number of APIs available for submitting events into Zenoss. For more information, see the *Zenoss Developer's Guide*.

Any ZenPacks you install may optionally include their own daemons. For more information, see *Zenoss Extended Monitoring*.

7.1.8. Creating Events Manually

You can manually create events. While this is not something you would do as part of normal Zenoss operation, it can be helpful when you are attempting to test mappings and transforms you have created.

7.1.8.1. Creating Events through the User Interface

To create events manually through the user interface:

1. Navigate to Events, and then select Add Event from the table menu.

 You also can create events from the Event Console.

The Add an Event dialog appears.

Figure 7.8. Add Event Dialog

- Complete the basic event fields. If you want any event class mappings to be applied to the event you are creating, you must select the blank Event Class (rather than the default /). Event class mappings are applied only for events that do not already have an event class.

7.1.8.2. Creating Events from the Command Line

To send events from the command line, use the `zensendevent` script, in this format:

```
zensendevent Options summary
```

Common options include:

- `-d DEVICE, --device=DEVICE`
- `-i IPADDRESS, --ipAddress=IPADDRESS`
- `-y EVENTKEY, --eventkey=EVENTKEY`
- `-p COMPONENT, --component=COMPONENT`
- `-k EVENTCLASSKEY, --eventclasskey=EVENTCLASSKEY`
- `-s SEVERITY, --severity=SEVERITY`
- `-c EVENTCLASS, --eventclass=EVENTCLASS`

7.1.8.2.1. Example

The following example shows how to use the `zensendevent` script to simulate a ping down event:

```
zensendevent -d router1.example.com -s Critical -c /Status/Ping "Router down"
```

7.1.9. Event Classes

Zenoss *event classes* are a simple organizational structure for the different types of events that Zenoss generates and receives. This organization is useful for driving alerting and reporting. You can, for example, create an alerting rule that sends you an email or pages you when the availability of a Web site or page is affected by filtering on the `/Status/Web` event class.

Following is a subset of the Zenoss default event classes. You can create additional event classes as needed.

- `/Status` - Used for events affecting availability.

- /Status/Ping - Ping up/down events
- /Status/Snmp - SNMP up/down events
- /Status/Web - Web site or page up/down events
- /Perf - Used for performance threshold events.
 - /Perf/CPU - CPU utilization events
 - /Perf/Memory - Memory utilization or paging events
 - /Perf/Interface - Network interface utilization events
 - /Perf/Filesystem - File system usage events
- /App - Application-related events.
- /Change - Events created when Zenoss finds changes in your environment.

7.1.9.1. Event Class zProperties

Just as device classes and devices have zProperties, so do event classes and event class mappings. zProperties are applied hierarchically, with the most specific zProperty being applied.

The following zProperties are available on event classes and even class mappings.

- **zEventAction** - How and where affected events are stored when they occur.
 - **status** - Active events table
 - **history** - Historical event table
 - **drop** - Events are not stored
- **zEventClearClasses** - Optional list of event class names whose active events will be cleared by clear events occurring in this class.
- **zEventSeverity** - The severity of affected events is changed to this value unless the Original value is used.

A good example of how Zenoss uses the event class zProperties is found in the /Change event class. Within the /Change event class' zProperties, zEventAction is set to drop and zEventSeverity is set to Info. This configuration causes all of the changes in your environment to be stored as info severity events in the history table.

For more information about event manipulation techniques, see the section titled "Mapping and Transformation."

7.1.10. Mapping and Transformation

Zenoss' event mapping and transformation system allows you to perform a wide range of operations, from altering the severity of certain events to altering nearly every field on an event, based on complex rules.

You cannot alter the following fields through event transformation. (This is because they are set after transformation has been performed.)

- evid
- firstTime
- lastTime
- count

The following illustration shows the path followed by an incoming event in the event mapping system.

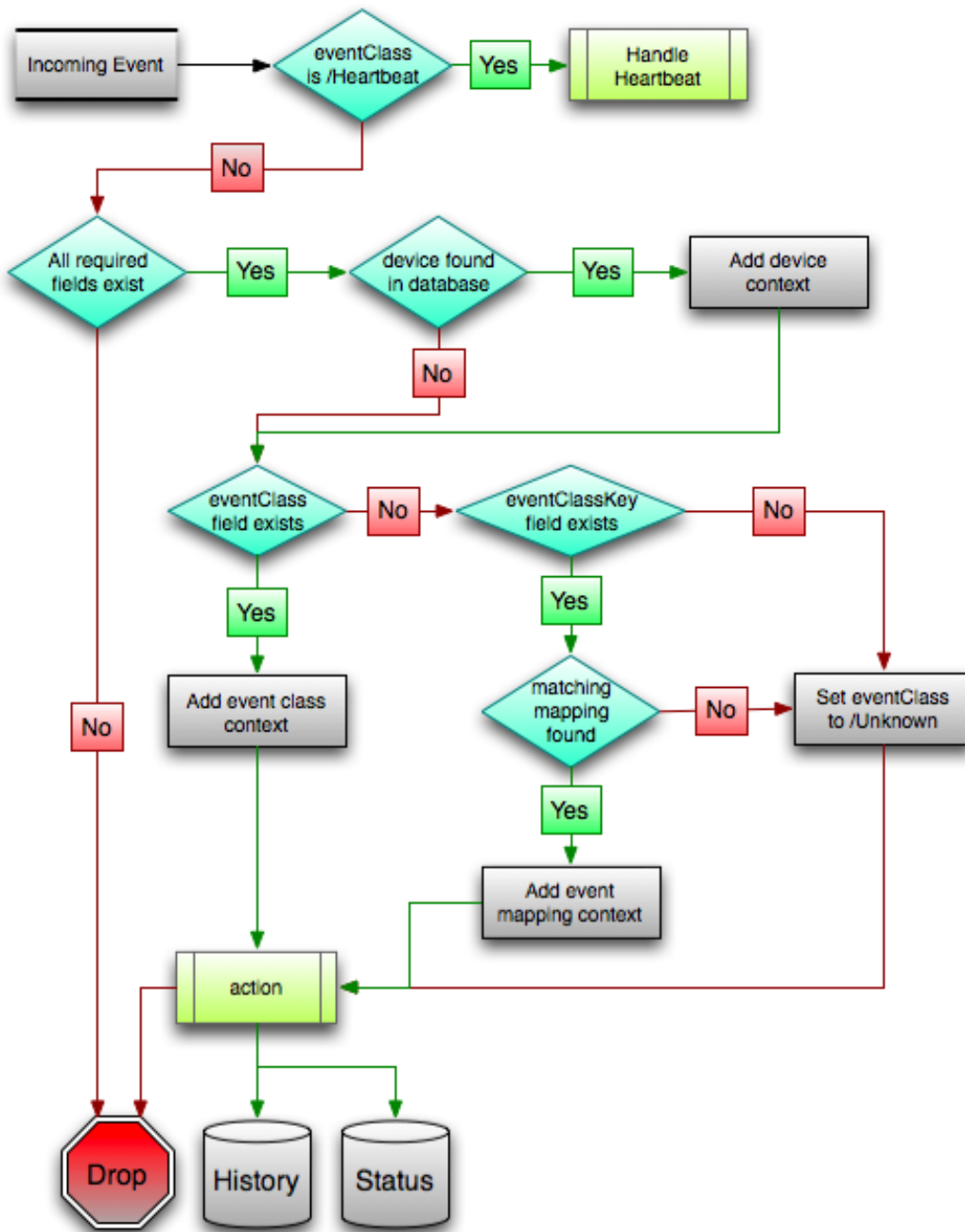


Figure 7.9. Event Processing

The mapping and transformation process begins with the "eventClass field exists" decision. This also is one of the more important differentiators in how you must handle a particular type of event.

7.1.10.1. Event Class Mappings

There are two primary ways to view the event class mappings that exist in the system. The first is to go to Events in the navigation menu and click on the Mappings tab. This allows you to see all event class mappings in a single location. The EventClass column shows which event class the mapping is in. The other way to view the existing event class mappings is to go to the Classes tab of any event class. This shows you only event class mappings related to the current event class.

You can create event class mappings directly from the event classes, but this requires that you know the event-ClassKey. A simpler way to create event class mappings is through the event console. Find an event that you want to match, select it, and then click **Classify**. Choose the event class that you want the event to be mapped

to, and then click **OK**. This will automatically create the event class mapping with the correct eventClassKey, and example text against which you potentially can develop your regular expression.

From the Edit tab of an event class mapping, you can control which events it will match, as well as other properties:

- **Name** - An identifier for this event class mapping. Not important for matching events.
- **Event Class Key** - Must match the incoming event's eventClassKey field for this mapping to be considered as a match for events.
- **Sequence** - Sequence number of this mapping, among mappings with an identical event class key property. Go to the Sequence tab to alter its position.
- **Rule** - Provides a programmatic secondary match requirement. It takes a Python expression. If the expression evaluates to True for an event, this mapping is applied.
- **Regex** - The regular expression match is used only in cases where the rule property is blank. It takes a Perl Compatible Regular Expression (PCRE). If the regex matches an event's message field, then this mapping is applied.
- **Transform** - Takes Python code that will be executed on the event only if it matches this mapping. For more details on transforms, see the section titled "Event Class Transform."
- **Explanation** - Free-form text field that can be used to add an explanation field to any event that matches this mapping.
- **Resolution** - Free-form text field that can be used to add a resolution field to any event that matches this mapping.

The sequence tab of an event class mapping allows you to handle situations where you need to provide more than one possible mapping for the same eventClassKey. In this case, the sequence is evaluated in ascending order until a full (rule or regex) match is found.

Mappings have the same zProperties as event classes. Any zProperty set locally on a mapping will override the same property set on the event class. This works in the same hierarchical, most specific match, concept that device class and device zProperties work.

When a captured event (see the section titled "Event Sources") occurs, it will not have an event class pre-defined. For this type of event, you must create an event class mapping if you want to affect the event. If a captured event occurs and none of the event class mappings in the system match it, its event class will be set to /Unknown, and it will retain all of the default properties that it began with.

The next step of evaluation for events without an event class is to check on the eventClassKey field. This is the first and most important field that controls which event class mapping the event will match. If the event has a blank eventClassKey, or its eventClassKey does not match any event class mappings in the system, the special "defaultmapping" eventClassKey is searched for instead. This provides for a way to map events even if they have a blank or unpredictable eventClassKey.

7.1.10.2. Event Class Transform

When a generated event occurs, it has an event class assigned to it. This causes the event class mapping step to be skipped. The only way to affect the fields of one of these events is through the event class' zProperties and transform.

To access the transform for an event class:

1. Navigate to the event class.
2. From the page menu, select More > Transform.
3. Enter information into the dialog (as Python code).

The objects available in this Python context are `evt` (the event); and, if the event matches a device that exists in the Zenoss database, a `device` object.

Example

The following example shows how you can validate that a device object exists before using it to drop events from a particular location.

```
if device and "Hawaii" in device.getLocationName():
    evt._action = "drop"
```

7.1.11. Event Life Cycle

In addition to some of the manual methods for getting events into the status or history tables, there are some automated processes that move events from status into history. The *event life cycle* is defined as all of the ways that events can get into the database, be moved within the database, and be deleted from the database.

The following illustration depicts the event life cycle.

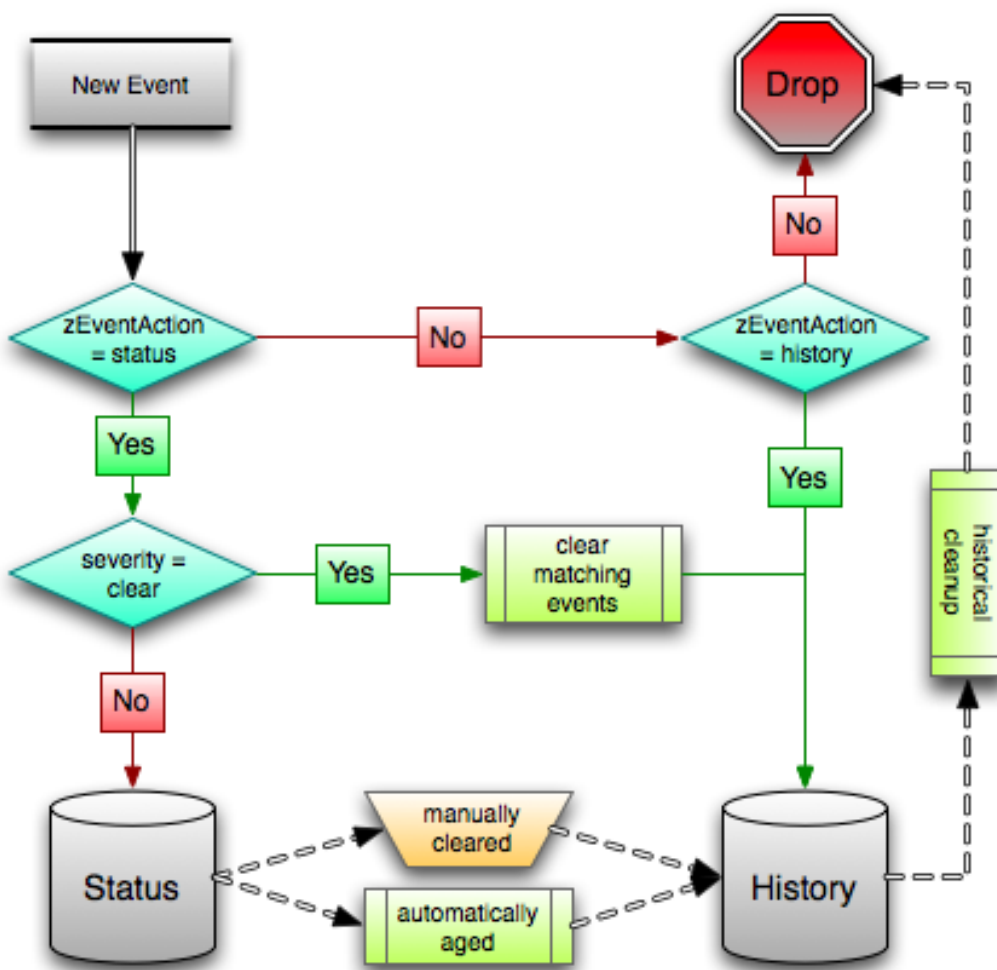


Figure 7.10. Event Life Cycle

7.1.11.1. Automatic Event Aging

From the event manager, you can set up automatic aging of certain events from the status table to the history table. This allows you to have lower severity events that do not reoccur for a specified length of time to be automatically archived to the history table.

Event manager properties that control this behavior are:

- **Event Aging Threshold (hours)** - By default, set to 4 hours.

- **Don't Age This Severity and Above** - By default, set to Warning.

With the default settings, Debug, Info, and Warning events that do not occur for four hours are automatically moved into the history table.

7.1.11.2. Automatic Historical Event Cleanup

You can set up automatic purging of events from the history table from the event manager. When events are purged from the history table, they can be recovered only from Zenoss backups. For this reason, the default setting is 0, which specifies that events are never automatically purged from history.

The event manager property that controls this behavior is Delete Historical Events Older Than (days).

7.1.12. Event Commands

Event commands allow Zenoss to run arbitrary shell commands when events occur that match pre-configured criteria. This allows almost any action to be taken in response to events that occur.

Common uses of event commands include:

- *Auto-remediation of events.* You can use SSH to remotely restart services on a UNIX system when they fail, or winexe to do the same for Windows services.
- *Integration with external systems.* This includes sending SNMP traps to other management systems, or opening tickets in your incident management system.
- *Extending alerting mechanisms.* Currently, Zenoss supports only email and pagers as alerting mechanisms "out of the box" through normal alerting rules. You could use event commands to alert through instant messaging systems, or by playing sounds.

The event commands that you configure are evaluated and executed by the `zenactions` daemon once each minute (just as in alerting rules).

7.1.12.1. Creating Event Commands

To create or edit event commands, go to the Commands tab of the event manager. From here, you can adjust these properties:

- **Enabled** - If set to True, then the command is evaluated and executed.
- **Default Command Timeout (secs)** - Length of time in seconds that Zenoss will wait for the commands to run that you specify in the Command and Clear Command fields. If the command takes longer than this, Zenoss kills it.
- **Delay (secs)** - Specifies the minimum age (in seconds) of an event before the command will be executed on it. This prevents commands from being run for flapping events.
- **Repeat Time (secs)** - If the command runs, then it will run again in the specified seconds if the triggering event is still active. Setting this value to 0 causes the command to be run only one time.
- **Command** - Specifies the command that will be executed in the shell when the criteria specified in the Where field match an event. This command is executed as the zenoss user, and uses TALES syntax for variable substitution. (For more information about TALES, see the appendix titled "TALES Expressions.") Available variables are `evt`, `device`, and `component`.
- **Clear Command** - Similar to the Command property. This is executed only when an event that originally matched the criteria is cleared.
- **Where** - Defines the criteria that an event must match to trigger this event command.

7.1.13. Capturing Email Messages as Zenoss Events

ZenMail and ZenPop allow you to capture email messages as events in Zenoss. This capability can be useful for situations in which embedded systems (such as WAPs, NAS devices, or RAID controllers) rely on email notification for events.

7.1.13.1. ZenMail

ZenMail serves as an SMTP server that you can bind to a specific TCP port. You can then configure your embedded system to send mail to the Zenoss server explicitly by using the Zenoss server's IP address as the relay.

ZenMail supports these configuration directives:

- `${ZENHOME}/bin/zenmail` (no arguments) - Default operation. Binds to port 25 on all ports and listens for email messages to arrive. Ignores the TO field in the email and uses the FROM address as the device IP address.
- `${ZENHOME}/bin/zenmail --listenPort` - Bind to the port provided. Useful in situations in which an SMTP server is already running on the Zenoss server and you do not want to interfere with the existing mail delivery system. Semantics are the same as the no argument version (FROM address is used as the device IP).

7.1.13.2. ZenPop

ZenPop allows you to retrieve event email from a POP server. ZenPop supports these configuration directives:

- `-usessl` - Issue the STARTTLS command to the POP server and attempt to transfer email messages using SSL encryption. This is required if retrieving mail from Google.
- `--nodelete` - Do not issue the DELE command after retrieving all messages. Typically this is used during initial testing so that you do not have to resend test messages to the POP account. Some email systems (such as Google) do not actually delete messages when the DELE command is issued.
- `--pophost` - The hostname or IP address of the POP server from which to retrieve messages.
- `--popport` - The TCP port the POP server listens on. Defaults to 110. Used in situations where the POP provider listens on another port (for example, Google on port 995).
- `--popuser` - The user name that contains email messages to retrieve.
- `--poppass` - The password to use for the user name provided.
- `--cycletime` - The time to sleep between polls. After all email is retrieved, ZenPop sleeps for this amount of time before waking up and attempting to pull new email.

7.1.13.3. Translating Message Elements to the Event

Zenoss translates various message elements to the event, as follows:

- **FROM Field** - If the FROM field is an IP address, then Zenoss associates the event with the device with the same IP address. If the FROM field is a fully qualified domain name, then Zenoss resolves it to an IP address, and then performs the device association using the resolved IP address. The resolution of hostname uses "A" records rather than "MX" records.
- **TO Field** - Zenoss ignores the TO field in the email message. ZenMail accepts email to any user and domain name combination. ZenPop also drops the TO field, and uses only the FROM field.
- **SUBJECT Field** - ZenMail and ZenPop use the SUBJECT as the event summary.
- **Message Body** - ZenMail and ZenPop use the first mime attachment as the event details. Zenoss ignores secondary message bodies (typically HTML-encoded versions of the message). It also ignores attachments (such as files).

Chapter 8. Production States and Maintenance Windows

8.1. About Production States and Maintenance Windows

Zenoss has the capability to support maintenance windows, or time periods, (scheduled or "on the fly") in which the monitoring and alerting rules are changed. A set of rules governing monitoring, display, and alerting is collectively defined as *production states*. When there are temporary changes in production state, and then a reversion to the original state, this is a *maintenance window*.

8.2. Production States

Production state determines whether a device is monitored, and can be used to control several elements of the event system, such as whether an event will produce a remote alert (email or page). Typically, devices start off their life in state "Pre-Production." In this state, devices are monitored by default, but no remote alerting occurs, and events are not shown on the Dashboard. Once a device is in full "Production" state, monitoring is occurring and remote alerts are sent. If service needs to be performed on a device, its state can be set to "Maintenance" to temporarily block any remote alerts.

There are three factors that determine how to choose a production state for a device:

1. Whether the device is being monitored.
2. Whether you want alerting to occur.
3. Whether the device appears on the dashboard.

Available production states are:

- Production - you want all three: monitoring, alerting and dashboard.
- Pre-Production - you may want monitoring but not alerting or the appearance of the device notices on the dashboard
- Test - you may want monitoring and alerting (sent to one email) and but not displaying device info on the dashboard.
- Maintenance - you want monitoring and collection to occur, and maybe or maybe not the device on the dashboard, just not alerting occurring
- Decommissioned - no monitoring, no dashboard, no alerting

8.2.1. Defining Production States for Devices

To set the production state for a device or group of devices, go to the Edit tab for the device or device group, and then select a production state from the options menu. When you add a device, its default production state is Production. If you change the production state for a hierarchy of devices, this state propagates down the hierarchy (unless you define an exception to the production state in the hierarchy).

8.3. Maintenance Windows

Maintenance windows allow scheduled production state changes of a device or all devices in a system, group, or location. Maintenance windows are defined on the Manage tab of these objects.

A Maintenance window has a Start Time, Duration, Repeat Cycle, and Start Production State. The Start Production State for the Maintenance Window is the state that the monitoring for the device (or group of devices) is in when the Maintenance window begins or opens. For example, if your devices are running in a production

state of “Production” (meaning you are monitoring and alerting on the devices normally) and the maintenance window opening time arrives, the production state changes to the maintenance window’s Start Production State.

For example, if the Start Production State is set to Maintenance, this means you want monitoring and data collection to continue to occur for the device, but you do not want alerts to occur or any warnings to appear on the dashboard. You can use this time to reboot the machine or make configuration changes that would normally create alerts and not have them actually send alerts. You can schedule a maintenance window or change the production state for the device manually at the time you want to make the changes. When the maintenance window closes, the devices change to the End Production State for the maintenance window. To set up a Maintenance Window, define the window such that when it’s time for the Maintenance Window to occur, the Start Production State should be Maintenance; and then when the Maintenance Window time frame expires, the Stop Production State should be production. (This means it returns to monitoring and alerting as normal.) This would save sending out known alerts as you rebooted or created other, known, alerting events.

8.3.1. Maintenance Window Events

When a maintenance window starts, an event is created with the following information:

- deupid - zenactions | *Monitor* | *MaintenanceWindowName* | *TargetOrganizerOrDevice*
- prodState - *StartProductionState*
- severity - Info
- summary/message - Maintenance window starting *MaintenanceWindow* for *Target*
- eventClass - /Status/Update
- eventClassKey - mw_change
- maintenance_devices - *Target*
- maintenance_window - *MaintenanceWindow*

When a maintenance window stops, an event is created with the following information:

- severity - Clear
- summary/message - Maintenance window stopping *MaintenanceWindow* for *Target*
- prodState - -99 (meaning "unknown.")

Maintenance window event auto-clear, meaning that stop events clear start events.

8.3.2. Creating and Using Maintenance Windows

You can create a maintenance window for an individual device or for any grouping of devices in any hierarchy you create and define. You can create these windows either for a single device or create a window per device class or system, and have the window propagate to all devices that fall below where you create the window.

To create a new maintenance window:

1. Navigate to the device or group of devices where you want to define the maintenance window. Open the page menu and select More and then Administration.

The Administration page appears.

2. From the Maintenance table menu, select Add Maint Window.

The Add Maintenance Window appears.

3. In the ID field, enter a name for the maintenance window, and then click **OK**.

The name appears in the Maintenance Window list.

4. To define the window, click the name of the Maintenance Window.

The Maintenance Window Status Tab appears.

5. Define the attributes for this Maintenance window.
 - **Name** - Name of the maintenance window.
 - **Enabled** - True or False as to whether you want this maintenance window active.
 - **Start** - The time for the window to become active.
 - **Duration** - The length of time for the window to be in effect starting from the Start time.
 - **Start Production State** - Defines the production state for the window before it opens.
6. Click **Save**.

Chapter 9. Organizers and Path Navigation

9.1. About Organizers and Path Navigation

You can group Zenoss objects, including devices, sub-systems, zProperties, and templates. A device, for example, can belong to multiple groupings, including groups, systems, locations, and device classes.

In the following illustration, the device `tilde.zenoss.loc` belongs to five different classifications. Any zProperties and monitoring settings for each of these groups are now applied to this device.

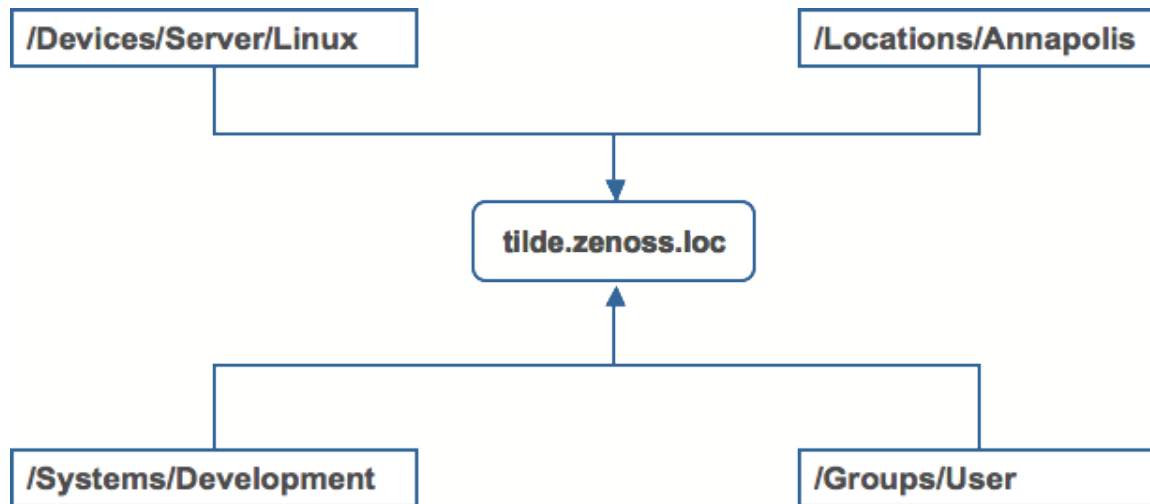


Figure 9.1. Device Groupings

Zenoss uses several organizers to classify and organize devices in the system.

- Class
- Systems
- Groups
- Locations

9.2. Classes

The most important organizers in Zenoss are classes, which comprise:

- Device classes
- Event classes
- Service classes
- Product classes

Templates and zProperties can be inherited based on class. These attributes can be overwritten further down the class hierarchy, all the way down to the individual component level. The class hierarchy includes all defined and standard classes and sub-classes.

The following procedures use device classes and sub-classes, but the same concepts apply to event classes, service classes, and product classes. When you add a device to Zenoss, you should (after providing the network name or IP address), at a minimum, specify its device class. Templates and zProperties can be set at any level in the device class hierarchy.

9.2.1. Viewing Device Classes

To see all of the devices in a device class, select Devices from the Navigation menu.

The Device Classes tab appears.

The screenshot shows the 'Device Class Tab' interface. On the left is a sidebar with navigation options: Main Views (Dashboard, Event Console, Device List, Network Map), Classes (Events, **Devices**, Services, Processes, Products), Browse By (Systems, Groups, Locations, Networks, Reports), and Management (Add Device, Mibs, Collectors, Settings, Event Manager). The main content area is titled '/Devices' and has tabs for 'Classes', 'Events', 'zProperties', and 'Templates'. The 'Classes' tab is active, showing a 'Summary' section with an 'event rainbow' for events: 4 (red), 36 (orange), 18 (yellow), 27 (blue), and 0 (green). Below this is a 'Sub-Devices' table with columns for Name, Subs, Devices, and Events. The table lists various device classes with their respective counts and event indicators. To the right of the table is a 'Devices' panel with a search bar and tabs for 'Name' and 'Events'.

| Name | Subs | Devices | Events |
|---|------|---------|--------|
| <input type="checkbox"/> Discovered | 1 | 0 | |
| <input type="checkbox"/> Network | 17 | 10 | |
| <input type="checkbox"/> Ping | 0 | 5 | |
| <input type="checkbox"/> Power | 2 | 2 | |
| <input type="checkbox"/> Printer | 2 | 1 | |
| <input type="checkbox"/> Server | 23 | 59 | |
| <input type="checkbox"/> Storage | 1 | 0 | |
| <input type="checkbox"/> Web | 2 | 3 | |

At the bottom of the 'Sub-Devices' table, there is a pagination control showing '1 of 8' items, a dropdown menu set to 'Discovered', and a 'Page Size' of 40.

Figure 9.2. Device Class Tab

The Device Class tab shows an "event rainbow" for that class level and a summary for the next level of class hierarchy, along with an indicator of whether there are any devices in any of the classes that have events associated with them.

9.2.2. Setting zProperties at the Class Level

To set zProperties at the Device Class Level:

1. Navigate to the Device Class tab for the class you where you want to set zProperties, and click the zProperties Tab.

The zProperties tab for this Device Class appears.

| Property | Value | Type | Path |
|-------------------------|----------------------|---------|------|
| zAxlPassword | ***** | string | / |
| zAxlUsername | administrator | string | / |
| zCollectorClientTimeout | 180 | int | / |
| zCollectorDecoding | latin-1 | string | / |
| zCollectorLogChanges | True | boolean | / |
| zCollectorPlugins | Edit | lines | / |
| zCommandCommandTimeout | 15.0 | float | / |
| zCommandCycleTime | 60 | int | / |
| zCommandExistenceTest | test -f %s | string | / |
| zCommandLoginTimeout | 10.0 | float | / |
| zCommandLoginTries | 1 | int | / |
| zCommandPassword | | string | / |
| zCommandPath | /opt/zenoss/libexec | string | / |
| zCommandPort | 22 | int | / |
| zCommandProtocol | ssh | string | / |
| zCommandSearchPath | | lines | / |
| zCommandUsername | | string | / |
| zDeviceTemplates | Device DnsMonitor | lines | / |

Figure 9.3. Device Class - zProperties Tab

2. Define any of the normal Device zProperties. These will apply to all devices in this class or added to this class unless overridden at a lower level in the hierarchy.

9.2.3. Defining and Applying Templates at the Class Level

To define Templates at the Device Class Level:

1. Navigate to the Device Class tab for the class you where you want to set Templates, and click the Templates Tab.

The Templates tab for this Device Class appears.

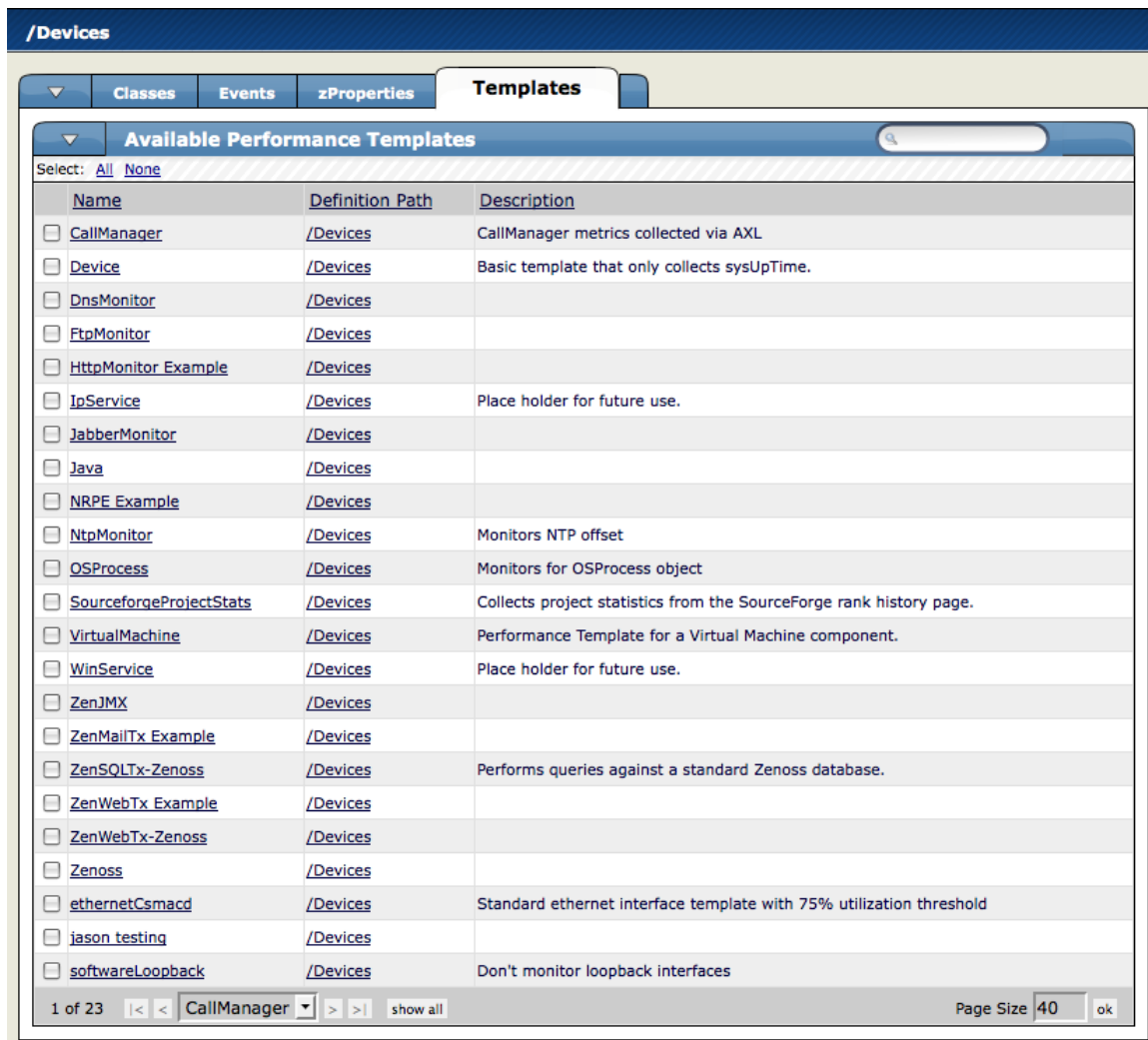


Figure 9.4. Device Class - Templates Tab

- Now you can define or bind any Device Template and they will apply to all devices in the class or added to this class unless over-ridden at a lower level in the hierarchy.

9.2.4. Creating Classes

To create a device class:

- From the class level where you want to add the organizer, navigate to the Classes tab.
- From the SubClasses table menu, select Add New Organizer, and then enter a name for the organizer.

To add devices to this device class:

- Navigate to the device list.
- Select the devices you want to add to the class.
- From the table menu, select Move to Class, and then select the class to which you want to move the device.

9.3. Systems

Systems are intended to follow virtual setups like you would have in a network setup or systems grouped by functionality.

9.3.1. Adding, Moving and Nesting Systems

To create a new system or sub-system:

1. From the Navigation menu on the left, under Browse by, select Systems.

The Sub-systems Status tab appears.

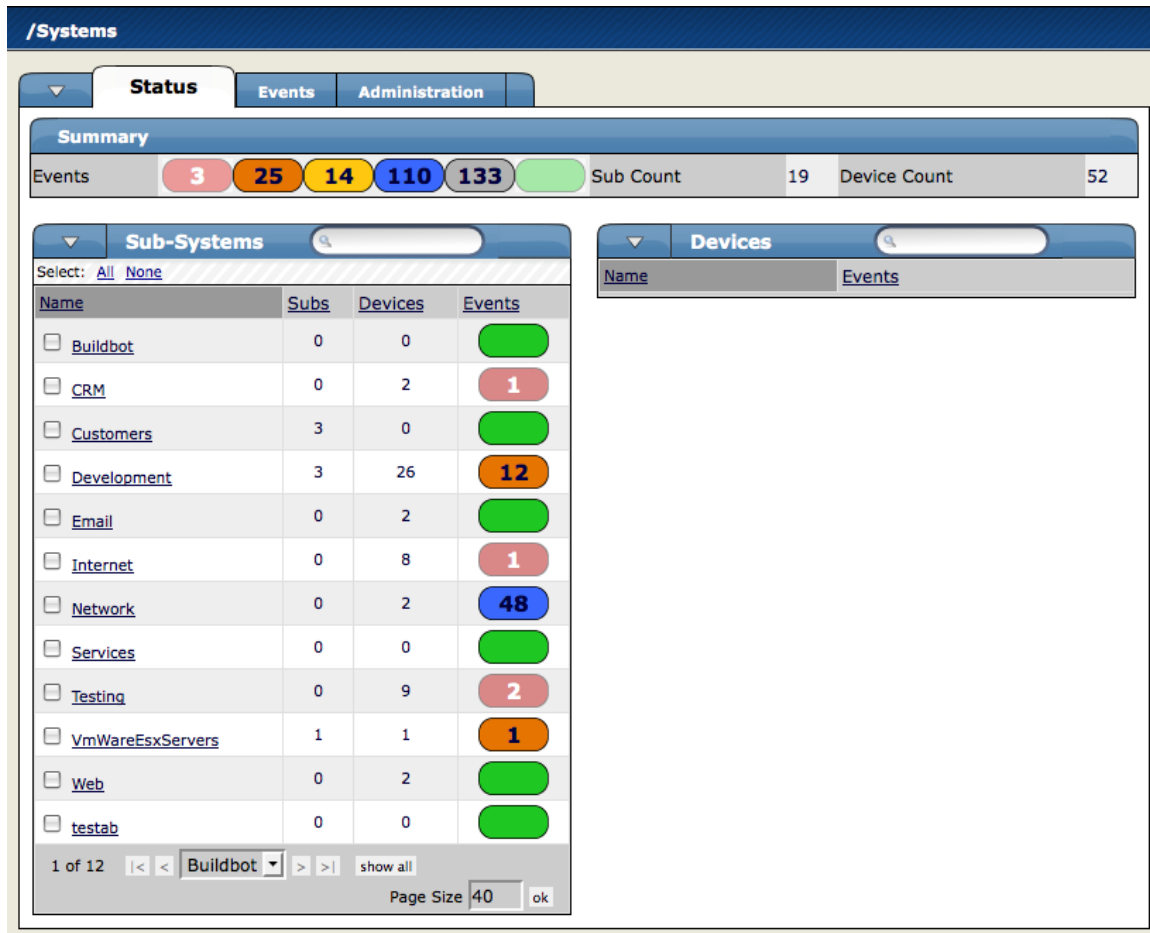


Figure 9.5. Sub-systems Status Menu

2. Open the Sub-Systems table menu and select the Add New Organizer option. The Add Organizer dialog appears.

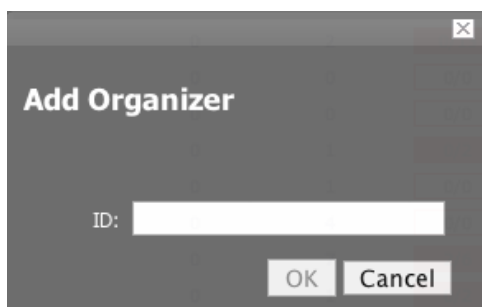


Figure 9.6. Add Organizer

3. In the ID field, Enter the name for the new Sub-system.
4. Click **OK**.

The new sub-system appears in the list.

9.3.1.1. Moving the Sub-System

To move the sub-system into another group or sub-group:

1. Select one or more systems that you want to move, and then open the Sub-Systems table menu to show the Subsystem options.
2. Select the Move Organizer option.

The Move Organizer dialog appears.

3. Select the location where you want to move this system.
4. Click **Move**.

The system is moved to the selected system. The attributes page for the newly selected system appears.

9.4. Groups

Using Groups is much like using systems only the intent is to use groups as functional divisions or even monitoring organizers to assign attributes to multiple objects with similar function or even among departmental lines. Note that groups do not appear on the Dashboard. Even though they do not appear on the dashboard, they still help to create additional organizers for monitoring or alerting.

9.4.1. Adding Groups

1. From the Navigation menu on the left, under Browse by, select Groups.
2. From the Settings tab, open the Sub-Groups table menu.
3. Select the Add New Organizer option.

The Add Organizer dialog appears.

4. In the ID field, Enter the name for the new Sub-Group.
5. Click **OK**. The new sub-group is added and appears in the list.

9.4.1.1. Moving Groups

To move the sub-group into another group or sub-group:

1. Select the group from the group list by clicking the check box next to the groups that you want to move and then open the Sub-Groups table menu to show the group options.
2. Select the Move Organizer option.

The Move Organizer dialog appears.

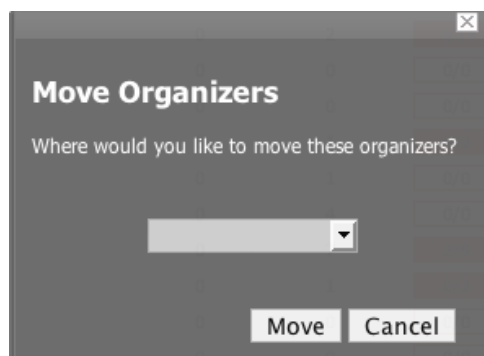


Figure 9.7. Move Organizer

3. From the pop-up menu, select where you want to move this group.
4. Click **Move**.

The group is moved to the selected group and the attributes page for the group where you moved the group appear.

9.5. Locations

Using Locations is much like using systems and groups only the intent is to use locations as logical groupings for physical locations of systems. They can be as general as city and state or as specific as rack or closet. It is completely customize-able. Note that locations also do not appear on the Dashboard. Even though they do not appear on the dashboard, they still help to create additional organizers for monitoring or alerting.

9.5.1. Integration with Google Maps

9.5.1.1. Overview

Zenoss can map locations by using a Google Maps mashup feature by setting the location's "Address" property to an address that Google Maps considers valid. The selected Location will appear on the map as a dot. The color of the dot represents the highest severity of any event on any device under that location. In addition, network connections that span locations will be represented on the map by lines also color-coordinated to the appropriate to the status of that connection.

To access the Google Map for a network location:

1. From the Navigation menu, click Locations.
2. Select the location you want to see the network map.
3. Click the Map tab.

The Network map is also one of the portlets you can add to the dashboard.

9.5.1.2. API Key

Before you can use the Google Maps feature, you must obtain or register for a Google Maps API key.

For Enterprise Implementations

To obtain a Google Maps API key, open a support case through the Zenoss customer support portal, at:

<http://support.zenoss.com>

Be sure to provide your Zenoss server key; this is required for Zenoss to generate the Google Maps API key. To find your Zenoss server key, select Settings, and then select the Versions tab.

For Core Implementations

You must register for a Google Maps API key. Free Google Maps API keys are linked to a base URL by which an application must be accessed for Google Maps to function.

To obtain a free Google Maps API key:

1. Point your browser to: <http://www.google.com/apis/maps/signup.html>
2. Fill in the URL by which you access your Zenoss Web interface. Include the port. For example, "http://localhost:8080".

Users accessing the Zenoss Web interface via a different URL than the one you specify here (for example, by IP instead of hostname) will not be able to use the Google Maps feature.

3. Agree to the terms and click **OK** to receive your API key. Copy it to the clipboard.

9.5.1.3. Setting an Address for a Location

1. From the left navigation menu, select Locations.

2. From the Summary table, next to Address, click Edit.

An Edit dialog slides down.

3. In the New Address field, enter a complete address that can, be resolved by Google Maps (if you are unsure, head to <http://maps.google.com> and see if it maps).
4. Click **Save**.

You now have created the address for the location that will appear on the map for this location. You must add at least one device to the Location for the dot to appear on the map.

9.5.1.4. Clearing the Google Maps Cache

Sometimes there are issues with drawing the maps and seeing the network status of locations or connections. Clearing the Geocode cache will solve these problems. To clear the Geocode cache, navigate to the Locations tab. From the page menu, select Clear Geocode Cache.

9.5.1.5. Network Links

If two devices in the same network are in different map-able locations, a line will be drawn on the map representing a network connection between the two. If there are multiple separate network connections between the same two locations, only one line will be drawn. The color of the line will represent the highest severity of any events affecting the connection. These are determined by:

- A ping down event on the device at either end of the connection; or
- Any event on the interface at either end of the connection.

9.5.1.5.1. zDrawMapLinks Property

Calculating network links on the fly is an expensive procedure. If you have a large number of Devices that have been assigned Locations, drawing those links on the map may take a long time. In order to save time, you can tell Zenoss not to attempt to draw links for specific networks (for example, a local network comprising many devices that you know does not span multiple Locations).

1. Navigate to the Network and click the "zProperties" tab.
2. Set the "zDrawMapLinks" property to "False."
3. Click "Save."

Like all zProperties, this setting will be inherited by all subnetworks. If you have few networks for which links would be drawn, it might be a good idea to set zDrawMapLinks to False on /Networks, and only set it to True on a network where you know a Location-spanning WAN connection exists.

9.5.1.6. Google Maps Example

This example will walk you through creating and displaying some Google map links of devices and sending a test event to see how the links are affected by changes in the system.

1. Navigate to /Network, click "zProperties" and set "zDrawMapLinks" to False. Save.
2. Navigate to /Locations and create two sub-Locations, called "New York" and "Los Angeles".
3. Go to the Status tab of each of the two new Locations. Set the "Address" property of each to "New York, NY" and "Los Angeles, CA" respectively.
4. Set the location of a device in Zenoss to /Locations/New York. Find another device on the same network and set its location to /Locations/Los Angeles.
5. Navigate to /Locations and click on the "Map" tab. Notice that both New York and Los Angeles are represented as dots on the map, but that there is no link drawn between the two.
6. Now navigate to the Network to which both devices are connected. Click the "zProperties" tab and set "zDrawMapLinks" to True. Save.

7. Navigate back to the "Map" tab of /Locations. Notice that a green line is now drawn between New York and Los Angeles.
8. Send an event to the device in /Locations/New York, with a severity of "Critical." Do not specify a component. Now refresh the /Locations "Map" tab. Notice that the New York dot has become red. Also notice that the link between New York and Los Angeles remains green.
9. Now navigate to the New York device's "OS" tab and determine the id of the component that is connected to the network shared with the Los Angeles device. Send another test event, but this time specify that component. Now refresh the /Locations "Map" tab and notice that the line linking the two locations has become red.

9.5.2. Adding, Moving, and Nesting Locations

To create a new Location or Sub-Location:

1. From the Navigation menu on the left, under Browse by, select Locations.

The Sub-locations Status tab appears.

2. Open the Sub-Locations table menu to show Sub-Locations options.
3. Select the Add New Organizer option.

The Add Organizer dialog appears.

4. In the ID field, Enter the name for the new Sub-location.
5. Click OK.

The new sub-location is added and appears in the list.

9.5.2.1. Moving Sub-locations

To move the sub-location into another location or sub-location:

1. Select the location from the location list by clicking the check box next to the locations that you want to move and then from the Sub-Locations table, select the Move Organizer option. The Move Organizer dialog appears.
2. From the pop-up menu, select where you want to move this location.
3. Click Move.

The location is moved to the selected location and the attributes page for the location where you moved the location appears.

9.6. Inheritance

Inheritance is defined by how many attributes are applied to a device at different points in the device hierarchy.

The following diagram shows an example of how and where zProperties can be set throughout the device class tree.

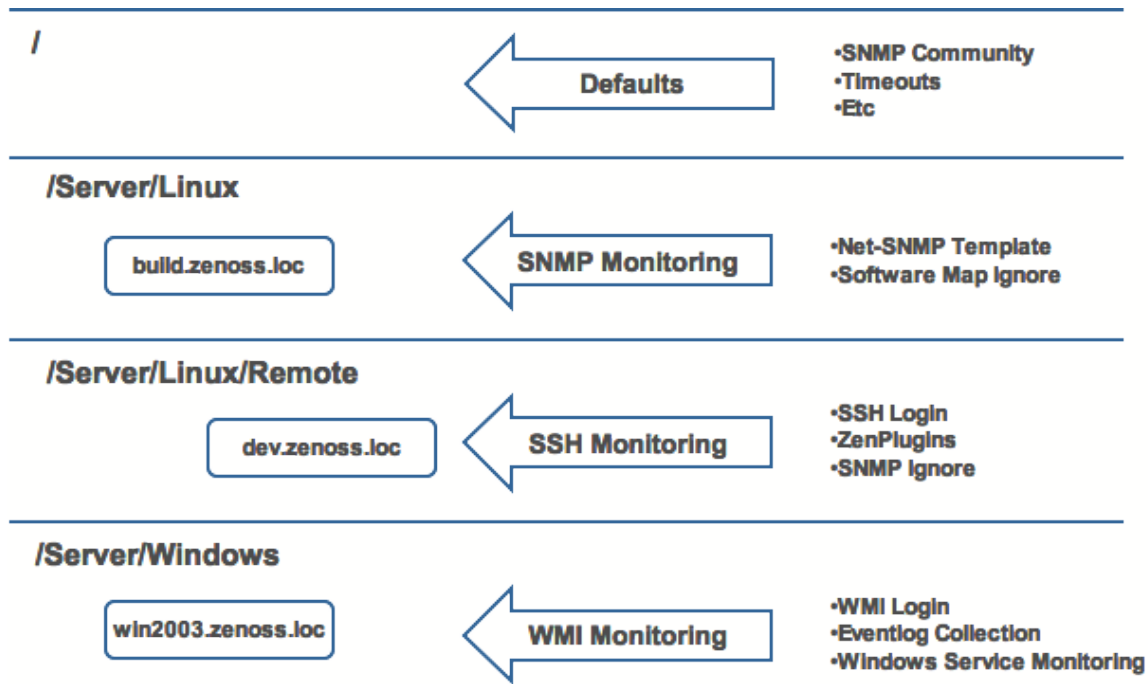


Figure 9.8. Zenoss Device Class Tree and Inheritance

In this example, you can see that the default properties can be set at the highest level (`/`), as you go further down the hierarchy, though, you can see that you can override any of the `zProperties` set at the root level. The next two lines show how the device tree further defines properties for Linux servers. If you wanted to set up and use SNMP monitoring for all Linux servers (inclusive of) `build.zenoss.loc`, you can change these properties at the `/Server/Linux` level. Now if you wanted to change how you collected information for remote Linux servers, you can create a sub-group within the `/Server/Linux` group called `/Server/Linux/Remote` and set these servers will use SSH monitoring and change the associated properties for that sub-group. Now also within the `/Server` group you can create another sub-group for Windows servers that change the `zProperties` specifically for WMI monitoring. All of these `zProperties` and groupings co-exist with any changes made lower in the hierarchy taking priority. It is very similar to a directory tree only you can place items in multiple organizers.

Chapter 10. User Commands

10.1. About User Commands

User commands allow users to execute arbitrary shell commands from within Zenoss. These commands can then be run manually against a single device or a group of devices. The user commands are executed on the Zenoss server, not on the remote device (unless the user command explicitly uses SSH to connect to the remote device.) User commands can be defined on any device class, system, group or location. They can also be defined globally from the Settings page under the Manage tab.

10.2. Defining User Commands

To define a user command:

1. From any device you have loaded into Zenoss, open the Device page menu, and then select More > Administration.

The Administration tab appears.

2. From Define Commands table menu, select Add User Command.

The Add User Command dialog appears.

A screenshot of a web-based dialog box titled "Add User Command". The dialog has a dark gray background with a light gray border. At the top, the title "Add User Command" is displayed in white. Below the title, there is a text input field labeled "Command Id:". The input field is empty and has a light gray border. At the bottom right of the dialog, there are two buttons: "OK" and "Cancel", both with a light gray background and dark gray text. A small "X" icon is visible in the top right corner of the dialog box.

Figure 10.1. Add User Command

3. In the Command Id field, enter a name for the command, and then click **OK**.

The Define Command page appears.

The screenshot shows a web interface for defining user commands. It features a tabbed interface with 'User Command' and 'Modifications' tabs. The 'Define Commands' section is active, showing a form with the following fields: 'Name' (a text input), 'Description' (a larger text area), 'Command' (a text input), and 'Confirm Your Password' (a text input). A 'Save' button is located at the bottom left of the form.

Figure 10.2. Define User Command

4. In the Description field, enter a brief description of what the command will do.
5. In the Command section, enter the TALEs expression-based command you want to run on the selected device or devices.
6. Enter your password for confirmation, and then click **Save**.

The Command is saved and added to the command menu so you can choose to run it at any time.

10.2.1. User Command Example: Echo Command

This section will walk you through creating an echo user command. You can see the use of TALEs expressions in the definition of this command.

1. Go to the Settings Manage tab.
2. Add a new command called “echoDevice”
3. Echo the name and IP of the device

```
echo name = ${here/id} ip = ${here/manageIp}
```

In a TALEs expression ‘here’ is the object that the expression is executed against. Some TALEs expressions in Zenoss have other variables like evt for event and dev or device for the device. See the TALEs Expression appendix for more information on the syntax of the various TALEs expressions.

4. Go to a device and run this command.
5. Now go back to the editing of this command and add some more information to the command.

```
echo name = ${here/id} ip = ${here/manageIp} hw = ${here/getHWproductName}
```

6. Now try running the command against a group of devices and see the command outputs.

Chapter 11. Managing Users

11.1. About Zenoss User Accounts

Each Zenoss user has a unique user ID, which allows you to assign group permissions and alerting rules that are unique to each user. Unique IDs also help ensure secure access to the system.

To create and manage user accounts, you must be logged in to the Zenoss admin account, or as a user with extended privileges.

Read the following sections to learn how to:

- Create user accounts
- Edit user accounts
- Work with user groups
- Assign users to roles
- Set up access control lists (ACLs)

11.2. Creating User Accounts

To create a user account:

1. From the Zenoss Dashboard Navigation menu, select Settings.
2. Click the Users tab.

The Users administration page appears.

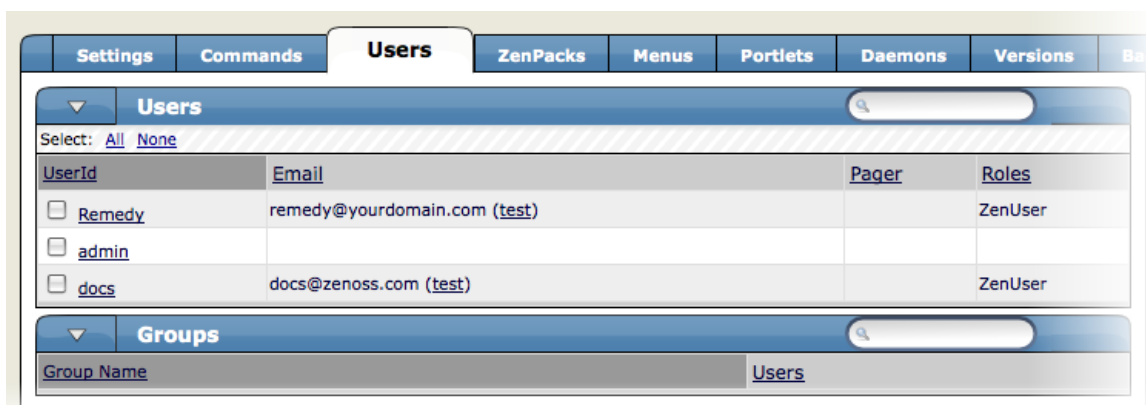


Figure 11.1. User Administration

3. From the Users table menu, select Add User.

The Add User dialog appears.

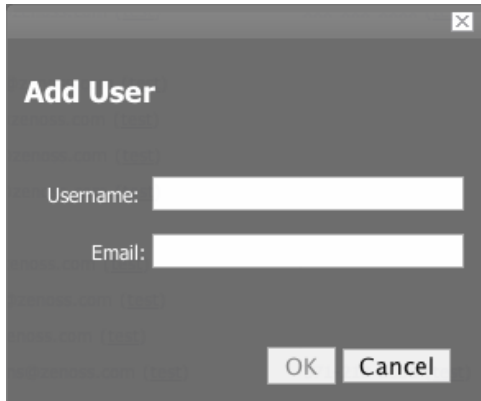
A screenshot of a software dialog box titled "Add User". The dialog box has a dark gray background and a light gray border. It contains two text input fields: "Username:" and "Email:". Below these fields are two buttons: "OK" and "Cancel". The dialog box is positioned in the upper left area of the page.

Figure 11.2. Add User

4. In the Username field, enter a unique name for the account.
5. In the Email field, enter the user account email address. Any alerts that you set up for this user will be send to this address.
6. Click **OK**.

The user appears in the User List.

After creating the account, edit the account to provide a password and additional user details. See the section titled "Editing User Accounts" for more information about setting user preferences.

11.3. Editing User Accounts

To access and edit user account information:

1. From the navigation menu, select Settings.
2. Click the Users Tab.

The Users Administration page appears.

3. Click the name of the user account you want to edit.

The individual User Administration page appears.

The screenshot shows the Zenoss user administration interface. At the top, there's a breadcrumb trail "/ZenUsers /admin". Below it are tabs: "Edit" (selected), "Administered Objects", "Event Views", and "Alerting Rules". A status bar indicates "State at time: 2009/10/06 12:33:53". The form contains several fields: "New Password" and "Confirm New Password" (text inputs), a "Reset Password" button, "Roles" (a dropdown menu showing "Manager", "ZenManager", and "ZenUser"), "Groups" (a text input), "Email" (text input with "ian@zenoss.com" and a "test" link), "Pager" (text input), "Default Page Size" (text input with "40"), "Default Admin Role" (dropdown menu with "ZenUser"), "Default Admin Level" (text input with "1"), and "Network Map Start Object" (text input). At the bottom, there's a red text prompt "Enter your password to confirm changes" and a "Save" button.

Figure 11.3. Edit User

4. Make changes to one or more settings:

- **New Password / Confirm New Password** - Enter and confirm a new password for the user.
- **Reset Password** - Facilitates user self-service by allowing a user to reset his password. Click to reset and email the new password to the email address associated with the user's account.
- **Roles** - Assign one or more roles (user privileges) to the user. To edit or assign roles, you must be a Zenoss Admin or be assigned the Manager role.

For more information about Zenoss user roles, and for a list of available roles and the privileges they provide, see the section titled Roles in this chapter.

- **Email** - Enter the user's email address. To verify that the address is valid, click the test link.
- **Pager** - Enter the user's pager number.
- **Default Page Size** - Controls how many entries (by default) appear in tables. Enter a value for the default page size. The default value is 40.
- **Default Admin Role** - Select the default role that this user will have for administered objects associated with him.

Enter your password, and then click **Save** to confirm and save your changes.

11.3.1. Associating Objects in Zenoss with Specific Users

You can associate any object in the Zenoss system with a particular user, for monitoring or reporting purposes. Once associated with a user, you can then assign the user a specific role that applies to his privileges with respect to that object.

For more information about object-specific roles, see the section titled Roles in this chapter.

To create an object association:

1. From the User Administration page, click the Administered Objects tab.

The list of administered objects appears.

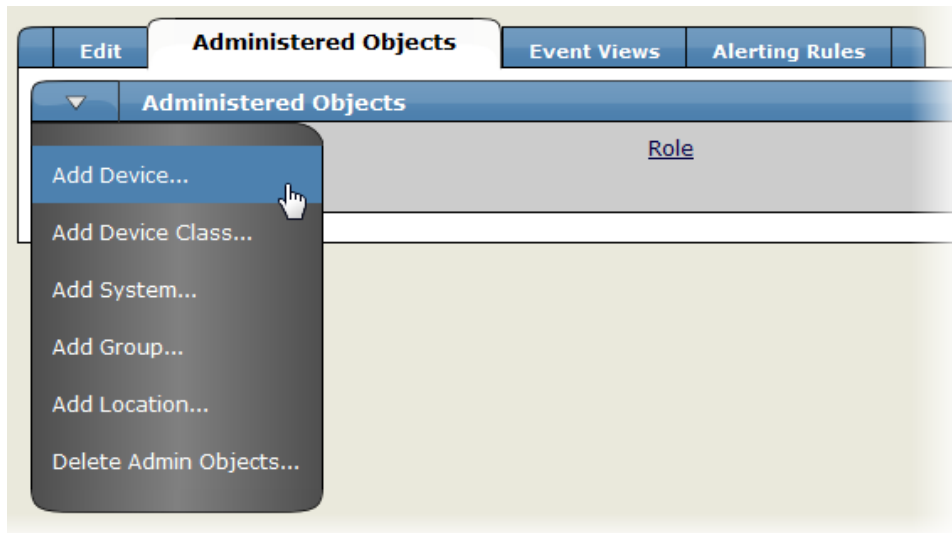


Figure 11.4. Administered Objects - Add Object

2. Select an object type from the Administered Objects table menu. You can add a Device, System, Group, or Location.

The Add Administered Device dialog appears.

3. Specify the component you want to add as an administered object, and then click OK.

The object appears in the Administered Devices list for the user.

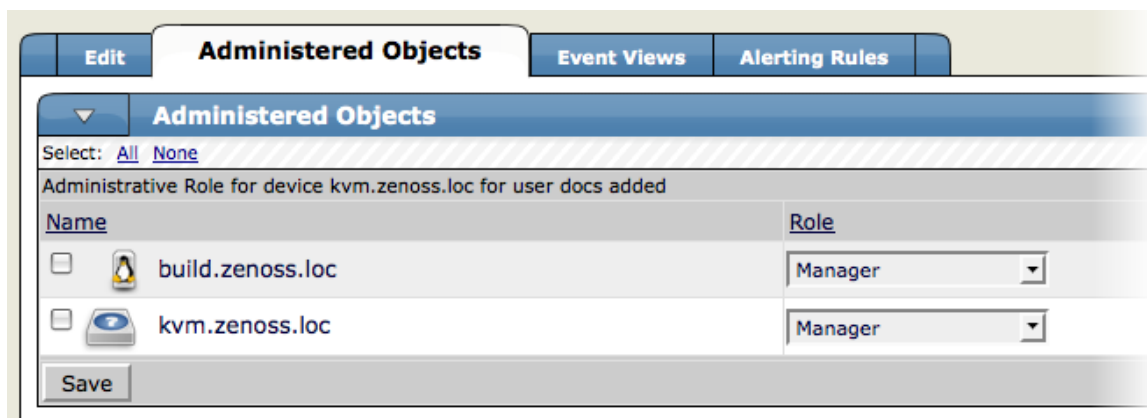



Figure 11.5. Administered Objects - Objects Added

4. Optionally, change the role that is associated for this user on this object.

 The default role assigned to the user for an administered object is specified by the Default Admin Role field on the Edit tab.

5. Click Save to save changes.

You can also set associated an object with a user by adding an administrator to the object. To do this:

1. Navigate to the object you want to add the user's list of administered objects.
2. From the page menu, select More, and then select Administration.

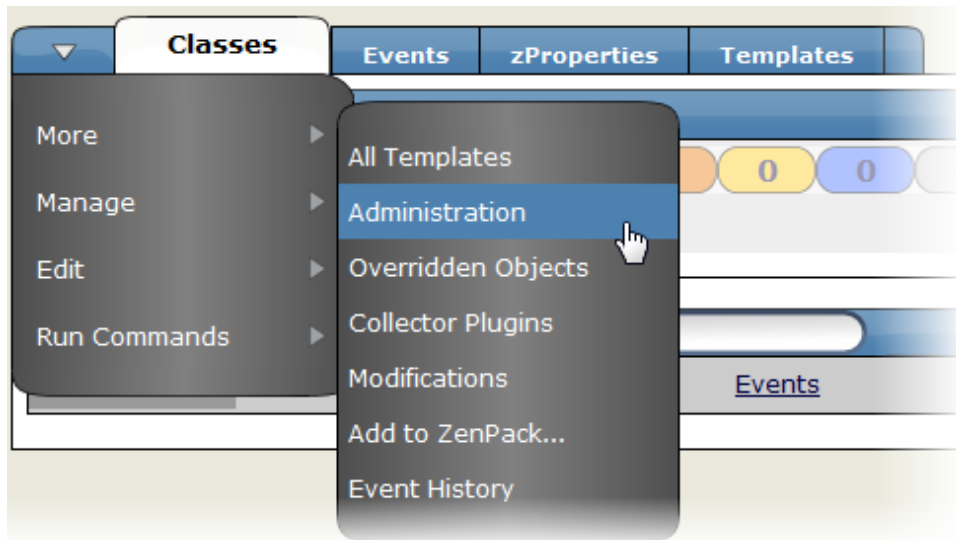


Figure 11.6. Administered Objects - Add Administrator

3. In the Administrators area, select Add Administrator from the table menu.

The Add Administrator dialog appears.

4. Select an administrator from the list, and then click **OK**.

The administrator appears in the object's Administrators list. The object is added to the user's Administered Objects list.

11.4. User Groups

Zenoss allows you to create user groups. By grouping users, you can aggregate rules and apply them across multiple user accounts.

Viewing User Groups

To view user groups, go to the Users tab of the Settings page. The groups area shows each user group and the users assigned to that group.

| Groups | |
|--|--|
| Group Name | Users |
| <input type="checkbox"/> CUSTOMER A ADMINS | |
| <input type="checkbox"/> Test | |
| <input type="checkbox"/> erictest | |
| <input type="checkbox"/> network | mturner |
| <input type="checkbox"/> testers | chris, drew, edahl, ian, jason, marc |

Figure 11.7. User Groups (Settings Page)

Creating User Groups

You can create Zenoss user groups to aggregate rules and apply them across multiple user accounts.

To create a user group:

1. From the Navigation menu, select Settings.
2. Click the Users tab.

The Users tab appears.

- From the Groups table menu, select Add New Group.

The Add New Group dialog appears.

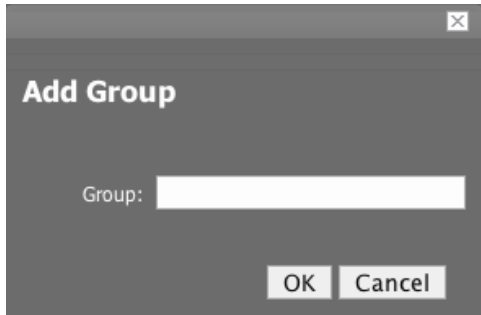
A screenshot of a dialog box titled "Add Group". It has a dark gray background and a light gray border. At the top right is a close button (X). Below the title is a label "Group:" followed by a text input field. At the bottom are two buttons: "OK" and "Cancel".

Figure 11.8. Add New User Group

- In the Group field, enter a name for this user group, and then click **OK**.

The group name appears in the Groups list.

- Click the name of the group you created.

The Edit tab for this group appears.

- From the Users In Group table menu, select Add User.

The Add User to Group dialog appears.

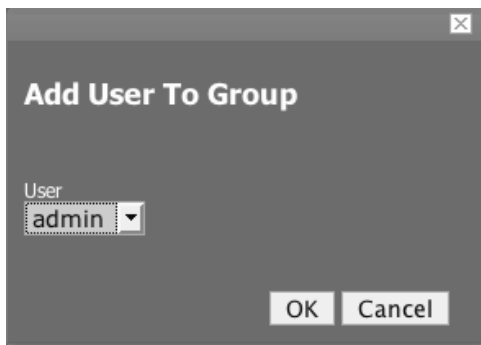
A screenshot of a dialog box titled "Add User To Group". It has a dark gray background and a light gray border. At the top right is a close button (X). Below the title is a label "User" followed by a dropdown menu showing "admin". At the bottom are two buttons: "OK" and "Cancel".

Figure 11.9. Add User to Group

- From the User list of selections, select the users you want to add to the group, and then click **OK**.

The user or users you select appear in the list of users for this group.

You also can choose administered objects and alerting rules for this user group. These alerting rules will apply to all users in the group. The user's original alerting rules and objects will also apply.

11.5. Roles

A role is a group of permissions that you can assign to users or groups in Zenoss.


The following table lists available Zenoss roles.

| Role | Definition |
|------------|--|
| ZenUser | Provides global read-only access to Zenoss objects. |
| ZenManager | Provides global read-write access to Zenoss objects. |

| Role | Definition |
|-------------|---|
| Manager | Provides global read-write access to Zenoss objects. Additionally provides read-write access to the underlying Zope object database. |
| ZenOperator | Installed by the ZenOperatorRole ZenPack. This ZenPack is available only to Zenoss Enterprise customers. The ZenOperator role provides users the ability to manage events. Combine the ZenOperator role with the ZenUser role to allow users read-only access to Zenoss, but also allow them to acknowledge events, move events to history, and add log messages to events. |

11.6. Device Access Control Lists

11.6.1. About Device Access Control Lists in Zenoss

 This feature is available only with Zenoss Enterprise.

The Device Access Control List (ACL) Enterprise ZenPack (ZenDeviceACL) adds fine-grained security controls to Zenoss. For example, this control can be used to give limited access to certain departments within a large organization or limit a customer to see only his own data. A user with limited access to objects also has a more limited view of features within the system. As an example, most global views, such as the network map, event console, and all types of class management, are not available. The Device List is available, as are the device organizers Systems, Groups, and Locations. A limited set of reports can also be accessed.

11.6.2. Key Elements

Following are key elements of device ACLs.

11.6.2.1. Permissions and Roles

Actions within Zenoss are assigned permissions. For instance to access the device edit screen you must have the "Change Device" permission. Permissions are not assigned directly to a user since this would be difficult to manage. Instead, permissions are granted to roles, which are then assigned to a user. A common example is the ZenUser role in Zenoss Core. Its primary permission is "View," which grants read-only access to all objects. ZenManagers have additional permissions such as "Change Device," which grants them access to the device edit screen. The Device ACL ZenPack has the role ZenRestrictedManager, which allows a more limited set of device edit functions. In Zenoss Core, when you assign a role to a user using the Roles field on the Edit tab, it is "global." When creating a restricted user you may not want to give that user any global role.

11.6.2.2. Administered Objects

Device ACLs provide limited control to various objects within the system. Administered objects are the same as the device organizers: Groups, Systems, and Locations and Devices. If access is granted to any device organizer, it flows down to all devices within that organizer. To assign access to objects for a restricted user, you must have the Manager or ZenManager group. Zenoss grants access to objects is granted using the "Administered Objects" tab of a user or user group. To limit access, you must not assign a "global" role to the user or group.

11.6.2.3. Users and Groups

Users and user groups work exactly as they would normally. See the section in the User Management section of this guide dealing with users and groups.

11.6.2.4. Assigning Administered Object Access

For each user or group there is a tab called "Administered Objects." The menu has an add item for each type of administered object. Adding an object will pull up a dialog box with live search on the given type of object. After an object has been added you can assign it a role. Roles can be different for each object so a user or group might have ZenUser on a particular device but ZenManager on a location organizer. If multiple roles are granted to a device though direct assignment and organizer assignment the resulting permissions will be additive. In the example above, if the device was within the organizer the user would inherit the ZenManager role on the device.

11.6.2.5. Portlet Access Control

Within Zenoss Core, portlet access can be controlled. This is important for Device ACLs.

11.6.3. Setup and Configuration Examples

Refer to the following examples for setup and configuration steps.

11.6.3.1. Restricted User with ZenUser Role

1. As admin or any user account with Manager or ZenManager role, create a user named acltest. Set a password for the user.
2. From the user's Edit tab, make sure that no role is assigned.
3. Select the user's "Administered Objects" tab.
4. From the menu, select the "Add Device..." item and add an existing device to that user.

The device's role will default to ZenUser.

5. Log out of your browser, or open a second browser and then log in as acltest.
6. Click on the "Device List".

You should see only the device you assigned to acltest.

7. Navigate to the device and notice that the Edit tab is not available. This is because you are in read-only mode for this device.

11.6.3.2. Restricted User with ZenManager Role

Following the example above:

1. Change the acltest user's role to "ZenManager" on the device. (You must to do this as a user with ZenManager global rights.)
2. Go back to the acltest user "Administered Objects" tab and set the role on the device to ZenManager.
3. As acltest, navigate back to the device. You now have access to the Edit tab.

11.6.3.3. Adding Device Organizers

1. Go to the Groups root and create a group called "RestrictGroup."
2. Go to the acltest user's Administered Objects tab and add the group to the user.
3. Logged in as acltest, notice that the Navigation menu has the Groups item. Group can be added to a user.
4. Place a device within this group and as acltest you should not only see the device within the group but also in the device list

11.6.3.4. Restricted User Organizer Management

1. Give the acltest user ZenManager on your restricted group.
2. As acltest, you can now add sub-organizers under the restricted group.

11.6.3.5. Viewing Events

A user in restricted mode does not have access to the global event console. The available events for the user can be seen under his organizers.

11.6.4. Detailed Restricted Screen Functionality

11.6.4.1. Dashboard

By default, the dashboard is configured with only three portlets:

- Object Watch List
- Device Issues
- Production State

These have content that will be restricted to objects for a given user.

11.6.4.2. Device List

The device list is automatically filtered to devices of a restricted user scoped to accessible devices. There are no menu items available.

11.6.4.3. Device Organizers

Device organizers control groups of devices for a restricted user. Every device added to the group will be accessible to the user. Permissions will be inherited down multiple tiers of a device organizer.

11.6.4.4. Reporting

Reports are limited to device reports and performance reports.

Chapter 12. Reporting

12.1. About Zenoss Reporting

Zenoss aggregates and reports, over time, on the data you set it up to monitor.

Zenoss reports are categorized as:

- Device reports
- Performance reports
- Graph and multi-graph reports
- Performance reports
- Other reports, which includes a notification schedule and heart beat report
- Custom device reports

To work with Zenoss reports, select Reports from the navigation area. The Reports list appears.

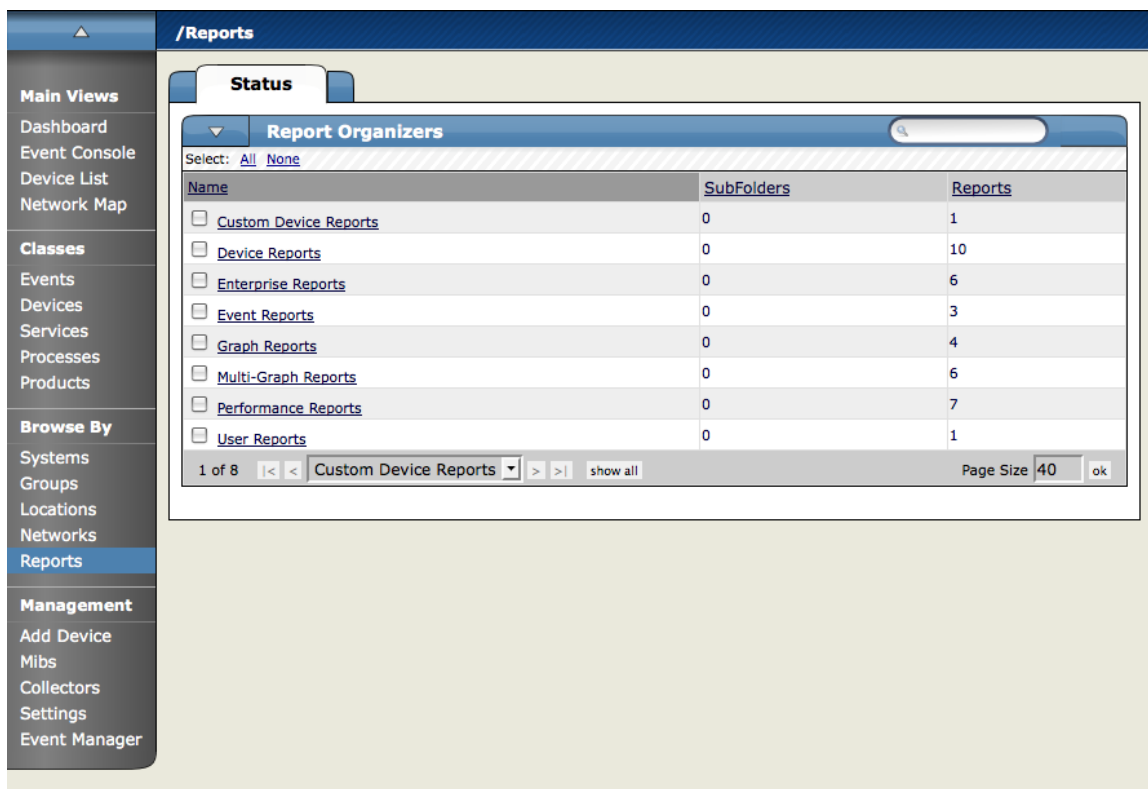


Figure 12.1. Reports List

12.2. Organizing Reports

You can organize reports in Zenoss in the same way you organize other elements in the system. You can create categories, sub-categories, and organizers, as well as report hierarchies and relationships.

12.3. Navigating and Sorting Report Results

Click a report column header to sort the report by that information category. You also can sort reports based on a key word that you enter in the Filter field.

12.4. Exporting Reports

You can export data from any Zenoss report, as comma-separated value (.csv) files.

To export report data, click Export All (located at the bottom of a report).

12.4.1. Advanced: Add An Export Button to a Report

You can edit any report to add an Export All button to export data.

The report format appears similar to the following:

```
<tal:block tal:define="
    objects python:here.ZenUsers.getAllThingsForReport();
    objects python: (hasattr(request, 'doExport') and list(objects)) or objects;
    tableName string: thisIsTheTableName;
    batch python:here.ZenTableManager.getBatch(
        tableName,objects, sortedHeader='getUserid');
    exportFields python:[
        'getUserid', 'id', 'delay', 'enabled', 'nextActiveNice',
        'nextDurationNice', 'repeatNice', 'where'];
">

<tal:block metal:use-macro="here/reportMacros/macros/exportableReport">
<tal:block metal:fill-slot="report"

<!-- Normal Report Markup Here -->

</tal:block>
</tal:block>
</tal:block>
```

Notable details in this example are as follows:

- The first definition is a call to a method that retrieves the objects for the report. This might be a list, tuple, or iterable class.
- The second `tal:define` line ensures that there is a list in the event being exported. (Do not do this unless doing an export; large reports may encounter performance issues if an iterable is unnecessarily converted to a list.)
- `tableName` is defined for use by the `getBatch()` call that follows.
- `exportFields` is a list of data to be included in the export. These can be attribute names or names of methods to call. See `DataRoot?.writeExportRows()` for more details on what can be included in this list.
- Within the `<tal:block metal:fill-slot="report"></tal:block>` block goes the report markup to use when not including the export functionality.
- If the Export All button does not appear to function, you may need to use `zenTableNavigation/macros/navtool` rather than `zenTableNavigation/macros/navbody` in your report. The former includes the `<form>` tag; the latter does not. If you are not providing a `<form>` tag then you need to use `navtool` so the export button is within a form.

12.5. Reports Included With Zenoss

Basic Zenoss reports are described in the following sections.

12.5.1. Device Reports

Device Reports aggregate and display data over many devices and device attributes.

- **All Monitored Components** – The All Monitored components shows all of the components currently being monitored. This is not all components in the system, only the ones on which Zenoss is currently collecting performance data. The information that appears in this report is: the device name, Component, the type of component (when available), the description, and the status of each device.

- **Model Collection Age** – The Model Collection age report lists devices that have not (but should have been) modeled during the past 48 hours. The information available in this report includes the device name, device class, the time when the device was first seen by Zenoss, the last time Zenoss Collected Data on this device, and any changes made to the configuration at that time.
- **Device Changes** - The Device Change report shows information about the history of any changes that Zenoss detects when modeling each device. This report only shows devices with changes. The information available in this report includes the device name, device class, the time when the device was first seen by Zenoss, the last time Zenoss collected data on this device, and any changes made to the configuration at that time.
- **Ping Status Issues** – The Ping Status Report shows the device name, the device class, the product name, the current state of the device, and the ping and SNMP status. The only devices that will show up here are devices that actually have or have had ping issues.
- **SNMP Status Issues** - The SNMP Status Report shows the device name, the device class, the product name, the current state of the device, and the ping and SNMP status. The only devices that will show up here are devices that actually have or have had SNMP issues.
- **New Devices** – The New Devices report shows devices that have been recently added. The report shows the device name, the device class, when the device was first seen, and the model collection age.

12.5.2. Event Reports

Event reporting gives you aggregate data about events, event mappings and event classes.

- **All Heartbeats** – The All Heartbeats Report shows all heartbeats for monitored devices. Heartbeats are reported by component and number of seconds.
- **All Event Classes** – The All Event Classes report shows all of the event classes that reside in the Zenoss system. It also breaks these classes down by sub-classes, the number of instances of that class in the system and the number of events in the system associated with each event class.
- **All Event Mappings** – The All Event Mappings shows all of the event mappings that you have created throughout the system. You can sort them by event class key, evaluation, or number of events for each event class.

12.5.3. Performance Reports

Performance Reporting allows you to roll up performance data across the Zenoss system.

- **Aggregate Reports** – The Aggregate reports shows the performance graphs for all of the devices in the system in graphical format. Common performance stats include PU Usage, Aggregate Free Memory, Aggregate Free Swap, and Network Output/Input. You can edit the graph parameters by clicking on the graph. You can change the Width, height, Min Y and Max Y axis. You can also specify which devices are included in the aggregate and the time span for the graph.
- **File System Utilization Report** – The File System Utilization Report shows the Total Bytes, Used Bytes, Free Bytes and % Utilization for each device. You can customize the report through the interface for such attributes and start/end Date and Summary type; either Average or Maximum.

This report uses data point aliases. (For more information about data point aliases, see the section titled "Data Point Aliases" in the chapter Core Monitoring.) To add data points to a report, add the alias, and then ensure the values return in the expected units.

| Alias | Expected Units |
|----------------------------|----------------|
| usedFilesystemSpace__bytes | bytes |

- **CPU Utilization Report** – The CPU Utilization report shows all of the Monitored Interfaces, the list of devices and the load average and % Utility. You can customize the report through the interface for such attributes and start/end Date and Summary type; either average or Maximum.


This report uses data point aliases. (For more information about data point aliases, see the section titled "Data Point Aliases" in the chapter Core Monitoring.) To add data points to a report, add the alias, and then ensure the values return in the expected units.

| Alias | Expected Units |
|-----------------|----------------|
| loadAverage5min | Processes |
| cpu_pct | Percent |

- **Threshold Summary** – The Threshold Summary Report identifies the devices that approach or exceed their thresholds and reports them in a list. You can see the device, the component, the event class, count, duration and percentage. You can filter this list by Event Class or to see all Event Classes, leave the event class Selection List blank. You can also change the start and end date for the reporting data.
- **Availability Report** – The Availability Report reports the percentage of time that a device or component is considered available. You can filter this report on device, component, event class, or severity. You also can further limit the time frame for the availability.

The value for a date range is calculated by summing the duration of all events of a particular class with a production state of "Production" and with a severity greater than or equal to a specified severity. This sum is then divided by the duration of the time range, and then subtracted from 1 and multiplied by 100 to get the percent available, as in:

$$(1 - \text{Total event down time}) / (\text{date range}) * 100$$

 Events that occur only once are not used in calculating device availability. Specifically, events whose firsttime and lasttime fields are the same are not used in the calculation. These could represent an event that occurs and is subsequently cleared by the next event, or an event that has happened only once in the specified date range.

- **Memory Utilization** – The Memory Utilization Report provides system-wide information about the memory usage for devices in the system. This report shows Total memory, Available memory, Cache memory, Buffered memory, and percent of memory utilized.

This report uses data point aliases. (For more information about data point aliases, see the section titled "Data Point Aliases" in the chapter Core Monitoring.) To add data points to a report, add the alias, and then ensure the values return in the expected units.

| Alias | Expected Units |
|------------------------|----------------|
| memoryAvailable__bytes | bytes |
| memoryBuffered__bytes | bytes |
| memoryCached__bytes | bytes |

- **Interface Utilization** - The Interface Utilization Report shows the traffic through all network interfaces monitored by the system. Columns included in the report are:
 - Device - Interface's device.
 - Interface - Interface.
 - Speed - Interface's rated bandwidth, in bits per second.
 - Input - Average traffic going out of the interface, in bits per second.
 - Output - Average traffic coming in to the interface, in bits per second.
 - Total - Total average traffic across the interface, in bits per second.
 - % Util - Average fraction of the interface's bandwidth consumed.

This report uses data point aliases. (For more information about data point aliases, see the section titled "Data Point Aliases" in the chapter Core Monitoring.) To add data points to a report, add the alias, and then ensure the values return in the expected units.

| Alias | Expected Units |
|---------------------|----------------|
| inputOctets__bytes | bytes/sec |
| outputOctets__bytes | bytes/sec |

12.5.4. User Reports

User Reports refer to report based on user account information and changes within the Zenoss system.

- **Notification Schedules** – The Notification Schedules Report shows all of the alerting rules and their associated details with each one.

12.6. Graph Reports

Graph Reports allow you to assemble graphs from specific Devices and Device Components into a single report. Click on Reports in the left navigation menu, then on the Graph Reports organizer to view or create Graph Reports. Graph Reports have a normal view which is similar to the graph views for Devices and Device Classes. Graph Reports also have a Printable view more appropriate for printing which can be brought up by clicking on the Printable button at the top of the report view.

Note that Graph Reports can only display graphs that already exist on Device or Components within Zenoss. You cannot define new graphs or alter existing graphs from within a Graph Report. If you need this type of functionality you probably want MultiGraph Reports instead.

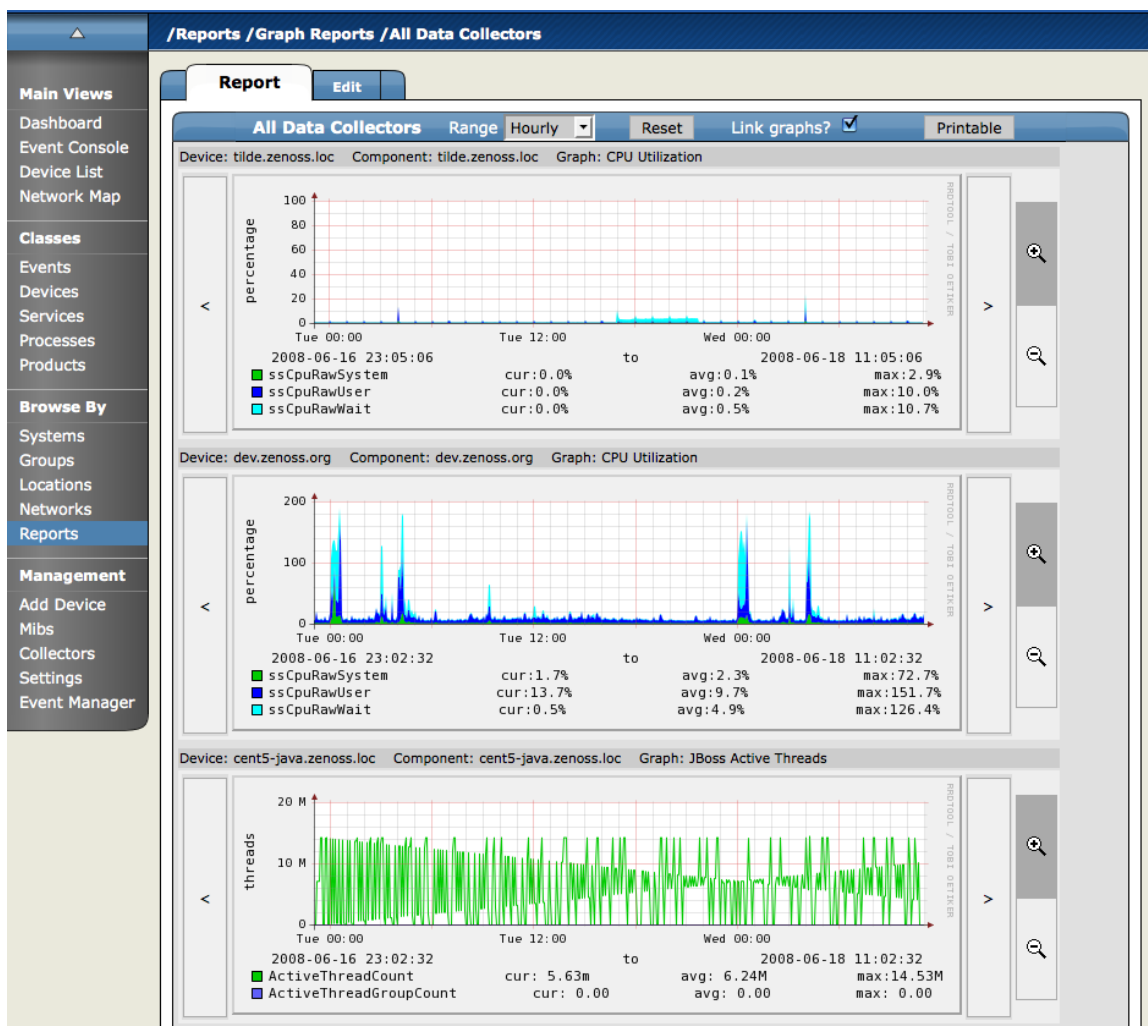


Figure 12.2. Graph Report

12.6.1. Creating a Graph Report

From within the Graph Reports organizer or a sub-organizer use the Add Graph Report menu item to create a new report. After entering a name for the new report you will be taken to the edit page. The fields for a Graph Report are:

- Name - The name of the report is displayed at the top of the report.
- Title - This description is displayed in the list of reports for the report organizer. It is also available for use in the comments.
- Number of Columns - This is the number of columns the graphs will be displayed in on the report.
- Comments - The comments are displayed at the top of the Printable version of the report. This is a TALEs evaluated string that may contain HTML formatting. The variables available to the TALEs expression are now (the current date and time) and report (the report object itself.)

12.6.2. Adding Graphs

The Add New Graph section of the Edit page allows you to add one or more graphs to the report. The first step is to select one or more Devices. The list of Devices can be narrowed by entering a search string to the left of the Filter button and pressing Return or clicking the Filter Button. When you select one or more Devices the Component list will display the names of all Components defined on at least one of the selected Devices. The Graph list will list the Graphs available on one or more of the listed Devices. When one or more Components are selected the Graph list will display Graphs from the selected Components rather than the selected Devices.

At any point you may select one or more Graphs and use the Add Graph to Report button. Zenoss steps through each selected Component (if any are selected) or Device (if no Components are selected) looking for graphs with the given names. Matching graphs are added to the Graph Report.

Graph Reports maintain a static list of graphs which does not change when graphs are added or deleted from Performance Templates. For example, take DeviceA which has only one graph called Graph1 and DeviceB which has two graphs named Graph1 and Graph2. On the Graph Report edit page if you selected DeviceA and DeviceB the list of graphs would include Graph1 and Graph2. Selecting both graphs and clicking the Add Graph to Report button would add three graphs to the report: DeviceA's Graph1, DeviceB's Graph1 and DeviceB's Graph2. If at some later date you created a Graph2 on one of DeviceA's Performance Templates it would not automatically appear on the Graph Report, you would have to edit the report to specifically add it. Similarly, if one of the graphs was removed from DeviceB's Template (or if DeviceB was deleted from Zenoss) you would need to manually remove them from the Graph Report.

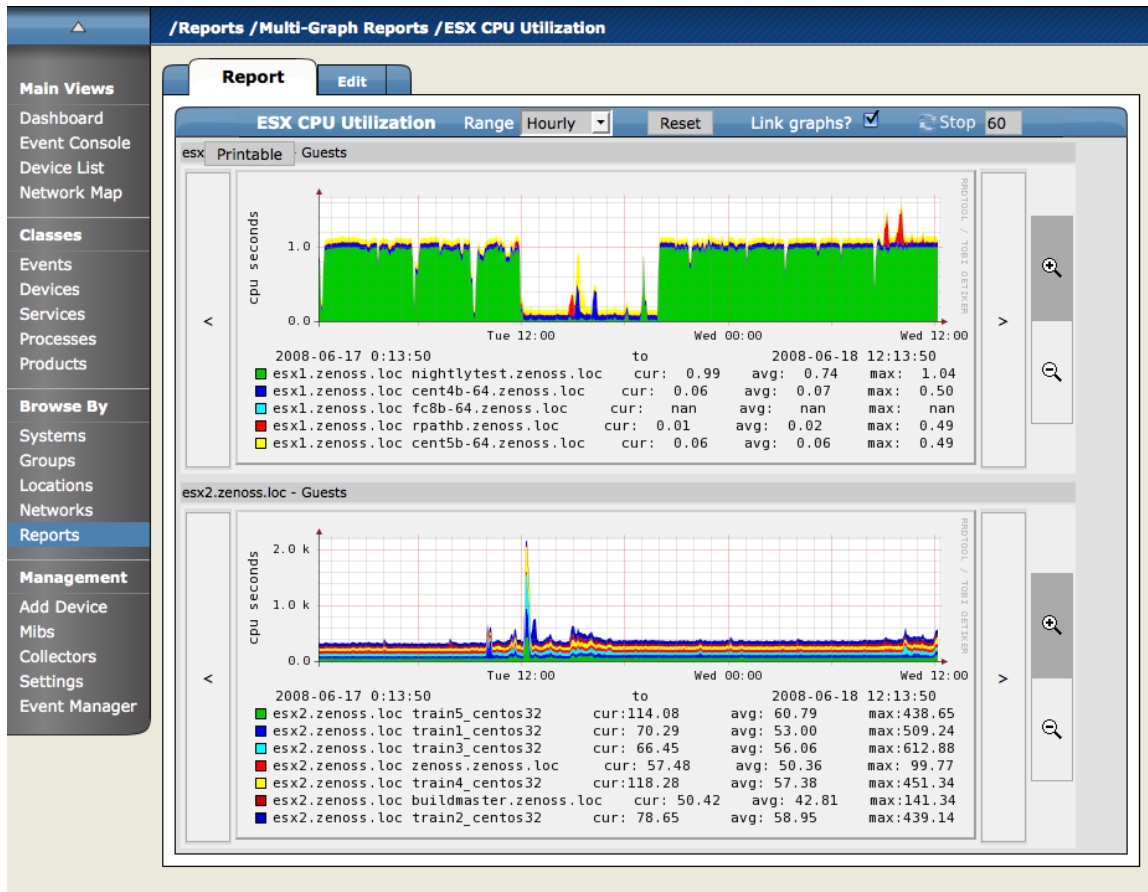


Figure 12.5. MultiGraph Report Graphs

12.7.1. Creating A MultiGraph Report

From within the MultiGraph Reports organizer or a sub-organizer use the Add MultiGraph Report menu item to create a new report. After entering a name for the new report you will be taken to the edit page. The fields for a MultiGraph Report are:

- Name - The name of the report is displayed at the top of the report.
- Title - This is displayed at the top of the printable version of the report.
- Number of Columns - The report will display the graphs in this many columns on the report.

Once you've created the MultiGraph Report there are three steps required to get graphs showing on the report:

1. Create a Collection which contains the Devices and/or Components you want to graph.
2. Create a Graph Definition that describes the graph(s) you want on the report.
3. Create a Graph Group which specifies the Collection and the Graph Definition you just created. The Method setting in the Graph Group lets you choose to have the graph drawn once for each Device/Component in the Collection or you can have the data from all the Devices/Components combined into a single graph.

These are just the minimal steps to getting a functional MultiGraph Report. You can create any number of Collections, Graph Definitions and Graph Groups in a single report. See the sections that follow for details on creating Collections, Graph Definitions and Graph Groups.

/Reports /Multi-Graph Reports /Network Bandwidth

Report **Edit**

State at time: 2008/06/18 12:08:38

Name: Network Bandwidth

Title:

Number of Columns: 1

Save

Collections

Select: All None

| Name | Number of Items |
|--|-----------------|
| <input type="checkbox"/> CatalystInterfaces | 9 |
| <input type="checkbox"/> GateBridgeInterface | 1 |
| <input type="checkbox"/> GateInterfaces | 2 |

Graph Definitions

Select: All None

| Name | Graph Points | Units | Height | Width |
|--|--|-------|--------|-------|
| <input type="checkbox"/> InputBandwidth | ifInOctets | | 100 | 500 |
| <input type="checkbox"/> OutputBandwidth | ifOutOctets | | 100 | 500 |
| <input type="checkbox"/> InterfaceHW | ifInOctetsPredicted, ifInOctets, ifOutOctetsPredicted, ifOutOctets | | 100 | 500 |

Graph Groups

Select: All None

| Seq | Name | Collection | Graph Definition |
|-----|---|--------------------|------------------|
| 0 | <input type="checkbox"/> CatOutputBandwidth | CatalystInterfaces | OutputBandwidth |
| 1 | <input type="checkbox"/> CatInputBandwidth | CatalystInterfaces | InputBandwidth |
| 2 | <input type="checkbox"/> GateBandwidth | GateInterfaces | InputBandwidth |

Figure 12.6. MultiGraph Report Edit Page

12.7.2. Collections

A Collection is a group of Devices and/or Components. A MultiGraph Report must have at least one Collection. Collections are listed in the Collections table on the report's Edit page. You can create a new Collection with the Add Collection menu item on that table then specifying a name in the dialog that appears.

A Collection consists of one or more Collection Items. A Collection Item is a list of Device Classes, Systems, Groups, Locations or specific Devices and Components that should be included in this Collection. You can create as many Collection Items of the various types as you wish within a single Collection. The controls for creating Collection Items are in the Add To Collection table of the Collection edit page. The Item Type menu lets you select one of the following:

- **Device Class/System/Group/Location** - Selecting one of these options reveals a list of all organizers of that type. You can select one or more of the organizers to include in the Collection. By selecting True for the Include Suborganizers field the Collection will also include all organizers recursively beneath the ones you selected. These Collection Items are dynamic - when devices are added or removed from these organizers they will appear or disappear from the report. Clicking the Add to Collection button creates a new Collection Item for each of the selected organizers.
- **Specific Device/Component** - Selecting this type reveals a list of all Devices in Zenoss. You can use the Filter field to narrow this list by entering a full or partial Device name. Selecting one or more Devices will display a list of Component names that apply to one or more of the selected Devices. If you do not select any Components and click the Add to Collection button then a new Collection Item is created for each selected Device.

Once you have specified an Item Type and made your selection click the Add to Collection button to create the new Collection Item. It will be added to the list of Collection Items at the end of the page. Collection items can be deleted or reordered within this list. Order of the Collection Items determines the order that the graphs are drawn in or the order that data is drawn on a combined graph. See the section on Graph Groups for more details.

/Reports /Multi-Graph Reports /Network Bandwidth /InputBandwidth

Graph Definition | Graph Custom Definition | Graph Commands

Graph Points

| Seq | Name | Type | Description |
|-----|-------------------------------------|-----------|-----------------------|
| 0 | <input type="checkbox"/> ifInOctets | DataPoint | ifInOctets_ifInOctets |

State at time: 2008/06/18 12:11:46

| | |
|-------------------------------------|----------------|
| Name | InputBandwidth |
| Height | 100 |
| Width | 500 |
| Units | |
| Logarithmic Scale | False |
| Base 1024 | False |
| Min Y | 0 |
| Max Y | -1 |
| Has Summary | True |
| <input type="button" value="Save"/> | |

Figure 12.8. MultiGraph Report Graph Definition

12.7.4. Graph Groups

Graph Groups are used to combine one Graph Definition with one Collection to produce graphs for the report. You must have at least one Graph Group or your report will have no graphs. You create a new Graph Group by selecting the Add Group menu item from the Graph Groups table menu. After entering a name for the Graph Group you are presented with the Graph Group edit page. There are 4 settings on this page:

- **Name** - This is used to identify the Graph Group on the MultiGraph Report page. It does not appear anywhere on the actual report.
- **Collection** - Select one of the Collections that have been defined for this report.
- **Graph Definition** - Select one of the Graph Definitions that have been defined for this report.
- **Method** - There are two options for applying the Graph Definition you selected to the Collection that you selected:
 - * **Separate graph for each device:** The Graph Definition will be used to draw one graph for each Device and Component listed in the Collection. The graphs will appear in the list in roughly the same order they are specified within the Collection.
 - * **All devices on a single graph:** This draws one graph with the data from all the Devices/Components on it.

The screenshot shows the 'Graph Group' configuration page in the Zenoss interface. The breadcrumb trail at the top is '/Reports /Multi-Graph Reports /Network Bandwidth /CatOutputBandwidth'. The left sidebar contains a navigation menu with sections: Main Views (Dashboard, Event Console, Device List, Network Map), Classes (Events, Devices, Services, Processes, Products), Browse By (Systems, Groups, Locations, Networks, Reports), and Management (Add Device, Mibs, Collectors, Settings, Event Manager). The 'Reports' item is highlighted. The main content area has a 'Graph Group' tab. Below it, a box titled 'State at time: 2008/06/18 12:12:49' contains a form with the following fields: Name (CatOutputBandwidth), Collection (CatalystInterfaces), Graph Definition (OutputBandwidth), and Method (All devices on single graph). A 'Save' button is located at the bottom of the form.

Figure 12.9. MultiGraph Report Graph Group

12.7.5. Graph Order

Graph Groups are drawn in the order they are listed on the MultiGraph Report edit page. You can change the order of the Graph Groups by editing the sequence number next to each then using the Re-Sequence Items menu item from that table's menu. If a Graph Group results in multiple graphs, the graphs are drawn in the order that the Collection Items are listed within the corresponding Collection. If a Collection Item specifies a Device organizer then the order of the Devices drawn from that Collection Item is indeterminate.

As with Graph Reports, if you have specified multiple columns for a report then the graphs are drawn left to right in that number of columns using as many rows as necessary.

12.8. Creating Custom Reports

Zenoss allows you to create custom reports. You can do this through the user interface, or by using the Zope Management Interface (ZMI).

12.8.1. Creating Custom Reports Using the ZMI

Zenoss reports are written in Python and templates are available through the ZMI.

To access the ZMI for a report, append the report page URL with `"/manage."`

12.8.2. Create A Custom Device Report: Example

The following example shows how to create a custom device report that will show device name, network address, and device serial number.

1. From the left navigation menu, select Reports.
2. From the Report Organizers list, click Custom Device Reports.

The list of custom device reports appears.

- From the Reports table menu, select Add Device Report.

The Add Report dialog appears.

- Enter a name for the report, and then click **OK**.

The Edit tab for the new report appears.

- Define report parameters, as follows.


- **Name** - Optionally edit the report name.
- **Title** - Enter a report title. This title shows in the report display, and is distinct from the report name.
- **Path** - Specify the path in the hierarchy where you want Zenoss to store the report.
- **Query** - Specify the actual query string for the report. If you want to limit the report to just those devices with a serial number, for example, you can set the Query value to:

```
here.hw.serialNumber != ""
```

- **Sort Column** - Specify the column on which you want to sort the report by default.
- **Sort Sense** - Specify the sense that the system uses to sort: asc (ascending sort) or desc (descending sort).
- **Columns** - Specify the data to be retrieved and displayed in the report.

For example:

- getId - Gets the name of any devices.
- getManagelp - Gets the IP addresses of the devices.
- getHWSerialNumber - Grabs serial numbers for the devices.

 For a complete list of valid options, refer to the information on Device schema in the appendix titled "TALES Expressions."

- **Column Names** - Optionally specify column names to make the column headers more descriptive.

For the columns shown in the previous example, you could use these column names:

- Device
- Address
- Serial #

- Click **Save**.
- Click the Report tab at the top of the page.

The new device report appears, showing the devices that meet the criteria you specified.

12.9. Using Reports to Help Troubleshoot Zenoss Daemons

This section will show you how to find and view certain reports that will aid you in troubleshooting Zenoss daemons.

From the Zenoss Web interface, navigate to Reports. Follow the path to the various reports listed below to see the reports. The troubleshooting items on the right give you clues as to what to expect to find in the various reports.

| Troubleshooting Items | Report Name | Where It Lives |
|--------------------------------|----------------------|-------------------------|
| zenmodeler issues | Model Collection Age | /Reports/Device Reports |
| Any internal Zenoss issues | All Heartbeats | /Reports/Event Reports/ |
| zendisc errors, adding devices | New Devices | /Reports/Device Reports |

| Troubleshooting Items | Report Name | Where It Lives |
|--|------------------------|-------------------------|
| Any alerting rule issues, will show all rules in the system | Notification Schedules | /Reports/User Reports |
| Summary of snmp status across the system including non-monitored | SNMP Status Issues | /Reports/Device Reports |
| Which devices are monitored and whether ping is turned on or off | Ping Status Issues | /Reports/Device Reports |

12.10. Scheduling Reports

By default, all reports in Zenoss run on demand and present information the moment you run the report. To schedule reports, you can use the `reportmail` tool. This allows you to pick a specific report and have its output emailed to a list of recipients on a pre-determined schedule. `reportmail` is a command line tool that is designed primarily to be run out of the UNIX crontab, allowing for flexible scheduling.

The following steps demonstrate how you would set up the standard Availability Report to be emailed to `<joe@example.com>` every Monday morning at 6 a.m.

1. Find the proper URL to the Availability Report by navigating to it in the Web interface and taking note of the URL in your browser.

2. Log in to your Zenoss server and switch to the zenoss user by running the command:


```
su - zenoss
```

3. Create a script that will invoke `reportmail` with the appropriate options with the following commands:

```
mkdir -p $ZENHOME/scripts
cat <<EOF > $ZENHOME/scripts/emailAvailabilityToJoe.sh
#!/bin/sh
REPORTS_URL="http://zenoss.example.com:8080/zport/dmd/Reports"


$ZENHOME/bin/reportmail \
--user=admin \
--password=zenoss \
--from="zenoss@example.com" \
--address="joe@example.com" \
--subject="Zenoss: Availability Report" \
--url="$REPORTS_URL/Performance Reports/Availability Report"
EOF

chmod 755 $ZENHOME/scripts/emailAvailabilityToJoe.sh
```

 You can run **reportmail --help** on your Zenoss server for additional help on `reportmail` options.

4. Run **crontab -e** to edit the zenoss user's crontab. You will be presented with an editor that allows you to set up scheduled commands.
5. Type a capital O to add a new line, and then enter the following job definition:

```
# Email availability report to Joe every Monday morning at 6am.  
0 6 * * 1 bash -lc '$ZENHOME/scripts/emailAvailabilityToJoe.sh'
```

 You can run **man 5 crontab** on your Zenoss server for more help on crontab formatting.

6. Type ESCAPE, and then :wq to save this crontab.

12.11. Advanced Zenoss Reports

The EnterpriseReports ZenPack, available only for Zenoss Enterprise, adds additional advanced reports to the standard core reports. See Enterprise Reports in *Zenoss Extended Monitoring* for more details.

Chapter 13. ZenPacks

13.1. About ZenPacks

A ZenPack is a package that adds new functionality to Zenoss. You can use ZenPacks to add action rules, event classes, event commands, user commands, service classes, data sources, graphs, performance templates, reports, model extensions, and product definitions. A ZenPack can also add daemons, and user interface features such as menus.

ZenPacks are a mechanism for extending and modifying Zenoss. This can be as simple as adding new device classes or performance templates, or as complex as extending the data model and providing new collection daemons. ZenPacks can be distributed for installation on other Zenoss systems. Simple ZenPacks can be created completely within the Zenoss user interface. More complex ZenPacks require development of scripts or daemons in Python or another programming language.

13.1.1. ZenPacks Provided by Zenoss

Zenoss offers a range of ZenPacks that add and extend functionality. These ZenPacks are grouped as "Core" ZenPacks (available to all Zenoss users) and "Commercial" ZenPacks, which are available only to Zenoss Enterprise.

The guide titled *Zenoss Extended Monitoring* provides detailed descriptions, installation information, and configuration details for each of the Zenoss ZenPacks.

13.2. Installing ZenPacks

ZenPacks are distributed as .egg files.

You can install ZenPacks from the command line on the Zenoss server, or from the Zenoss user interface.

13.2.1. Installing from the Command Line

The following ZenPack command can be used from the command line to install ZenPack files. After installing or updating ZenPacks you need to restart:

```
zenpack --install <filename>
zenoss restart
```

If you have the source code for the ZenPack you can install directly from that rather than from a .egg file. The command is the same, you just specify the directory containing the source code. This copies the source code into either \$ZENHOME/ZenPacks (for newer egg ZenPacks) or \$ZENHOME/Products (for older style ZenPacks.)

```
zenpack --install <directoryname>
zenoss restart
```

If you are developing a ZenPack you usually will want to maintain your source code outside of \$ZENHOME/ZenPacks or \$ZENHOME/Products. This is advisable for two reasons. First, if you issue a **zenpack --remove** command it will delete your code from either of those two locations and you would lose your files unless you had them backed up elsewhere. Second, if you are maintaining your source code in a version control system it is frequently more convenient to have the files reside elsewhere on the file system. Using the **--link** option you can install the ZenPack but have Zenoss use your code from its current location. Instead of installing your code in \$ZENHOME/ZenPacks or \$ZENHOME/Products Zenoss will create a link in one of those locations that points to your source code directory.

```
zenpack --link --install <directoryname>
zenoss restart
```

13.2.2. Installing from the User Interface

You can upload and install a ZenPack .egg file via the user interface. To do this:

1. From the navigation menu, select Settings.
2. Select the ZenPacks tab.
3. From the Loaded ZenPacks table menu, select Install ZenPack.

The Install ZenPack dialog appears.

4. Browse to and select the `.egg` file you want to install, and then click **OK**.

The file is uploaded to the Zenoss server and installed.

After installing the ZenPack, you should restart Zenoss.

13.2.3. Installing All Core ZenPacks via RPM

The Zenoss Core ZenPacks, along with third party ZenPacks, are available for download individually from:

<http://community.zenoss.org/community/zenpacks>

Also on that page is a link to download an RPM that includes the most popular Core ZenPacks. To install via the Core ZenPacks RPM follow these steps:

1. Download the appropriate file from the Zenoss ZenPacks download area specific to your version.
2. Make sure ZEO is running (as the zenoss user):

```
zeoctl start
```

3. Install the rpm (as root user):

```
rpm -ihv <rpm file>
```

4. Restart Zope and ZenHub:

```
zopectl restart  
zenhub restart
```

13.3. Creating ZenPacks

Say you have developed a performance template for a new piece of hardware. You have created data sources for the OID's you think are worth monitoring, thresholds to make sure some of these values stay within reasonable limits, and several graph definitions to show this data graphically. Perhaps you also have created a new device class for this hardware. You can create a ZenPack to easily distribute your performance template and device class to other Zenoss administrators. This ZenPack can be entirely created from within the Zenoss user interface.

For another example, say you want to monitor a new piece of software running on one of your servers. You would like to monitor several performance metrics of this software, but they are available only via a programmatic API provided with the software. You could develop a new collector daemon to gather data via this API and provide it back to Zenoss. You might also create a new type of data source to provide configuration data for the new collector. Obviously this effort would require development skills and intimate knowledge of Zenoss not necessary for the previous example, but this functionality can be distributed as a ZenPack.

Use the following instructions and guidelines to create a ZenPack.

When logged into Zenoss as an Administrator click on the Setting link and then on the ZenPacks tab. Select the "Create a ZenPack..." menu item. You will get a dialog asking for a name for your new ZenPack. The name must be of the form *ZenPacks.Organization.Identifier*, where *Organization* is a name that identifies you or your organization and *Identifier* is a string that represents the intent of your ZenPack. For example, *ZenPacks.zenoss.HttpMonitor* was created by Zenoss to help monitor HTTP sites. Once you have entered a name, click **Save**. This creates both the ZenPack object in the database as well as a new directory in the file system `$ZENHOME/ZenPacks/YourZenPackID`.

Many types of objects can be added to a ZenPack via the user interface. Some examples are:

- Device Classes
- Event Classes
- Event Mappings
- User Commands
- Event Commands
- Service Classes
- Device Organizers
- Performance Templates

Devices are the conspicuous omission from this list. Any individual Device is usually specific to a particular site and therefore not likely to be useful to other Zenoss users.

To add one of these database objects to a ZenPack navigate to that object and use the "Add to ZenPack..." menu item. Zenoss will present a dialog which lists all installed ZenPacks. Select the ZenPack to which you want to add this object and click the Add button. To view the objects that are part of a ZenPack navigate to the Settings page then the ZenPacks tab. Click on the name of the ZenPack and you will see a page that lists both the files and the objects that are part of this ZenPack. You can remove objects from the ZenPack by selecting the checkboxes next to them and using the "Delete from ZenPack..." menu item.

ZenPacks can contain items that are not ZEO database items, such as new daemons, Data Source types, skins, etc. These are added to a ZenPack by placing them in the appropriate subdirectory within the ZenPack's directory. See the Core ZenPacks at <http://community.zenoss.org/community/zenpacks> for examples of how to incorporate such items into your ZenPack. Further information regarding ZenPack development is available in the Zenoss Developers Guide.

Discussion regarding development of ZenPacks takes place on the zenoss-dev mailing list and forums:

<http://community.zenoss.org/community/forums>

13.3.1. Packaging and Distributing Your ZenPack

ZenPacks are distributed as .egg files. To create the installable .egg file for a ZenPack: use the "Export ZenPack..." menu item in the page menu when viewing a ZenPack. The dialog that follows has two options. The first option simply exports the ZenPack to a file named *ZenPackID*.egg in the `$ZENHOME/exports` directory on the Zenoss server. The second option does the same, but then downloads the exported file to your browser. Other Zenoss administrators can install this exported .egg file as described in the Installing ZenPacks section.

For information on developing ZenPacks, go to:

http://community.zenoss.org/community/developers/zenpack_development

13.4. Removing ZenPacks

Warning: Removing a ZenPack can have unexpected consequences. For example, removing a ZenPack that installed a device class will remove both the device class and all devices within that class. Also, before removing a ZenPack you should delete any Data Source of a type provided by the ZenPack. You should always perform a backup of your Zenoss data before removing a ZenPack. See Section 21.1 Backup and Restore for information on how to backup your Zenoss data.

Chapter 14. General Administration and Settings

Read the following sections to learn more about performing general administration tasks in Zenoss, including:

- Sending alerts to email recipients or pagers
- Adjusting event manager settings
- Setting portlet permissions
- Performing backup and recovery tasks
- Viewing, starting, and stopping jobs
- Tuning and maintaining the system

14.1. Email and Pager Settings

You can send Zenoss alerts to users via email (SMTP) or pager (SNPP). Many operating systems include an SMTP server (such as Sendmail or Postfix) with their distributions. If your OS does not include a mail server, you must install one or specify a separate SMTP server in the Zenoss settings. Many pagers can accept messages via email, but Zenoss also provides the option of sending pages via SNPP if you specify an SNPP server in Zenoss settings.

14.1.1. Setting SMTP and SNPP Information

To edit Zenoss SMTP and SNPP settings:

1. While logged in to a user account with management privileges, from the Navigation menu, click Settings.

The Settings Tab appears.

Settings | Commands | Users | ZenPacks | Menus | Portlets | Daemons | Versions | Backups

State at time: 2008/06/18 13:43:14

| | |
|--|--|
| SMTP Host | localhost |
| SMTP Port (usually 25) | 25 |
| SMTP Username (blank for none) | docs |
| SMTP Password (blank for none) | ***** |
| From Address for Emails | zenosst@zenoss.com |
| Use TLS? | <input type="checkbox"/> |
| Page Command | \$ZENHOME/bin/zensnpp I |
| Dashboard Production State Threshold | 1000 |
| Dashboard Priority Threshold | 2 |
| State Conversions | Production: 1000 Pre-Production: 500 Test: 400 Maintenance: 300 Decommissioned: -1 |
| Priority Conversions | Highest: 5 High: 4 Normal: 3 Low: 2 Lowest: 1 Trivial: 0 |
| Administrative Roles | Administrator Analyst Engineer Tester |
| Google Maps API Key Help | zenoss |

Figure 14.1. Zenoss Settings - Settings Tab

- Change the following SMTP settings, as necessary:

| Field | Description |
|-------------------------|---|
| SMTP Host | Set the SMTP Host value to your corporate email server. |
| SMTP Port | Usually port 25. |
| SMTP Username | Leave this field blank. |
| SMTP Password | Leave this field blank. |
| From Address for Emails | Enter a value if you want email to come from a specific email address. |
| Use TLS? | Select this option if you use transport layer security for your email alerts. |

Table 14.1. SMTP Options

- Enter a Page Command as necessary if you are using Zenoss to send pages. The pageCommand variable enables Zenoss to execute the pageCommand when a page is sent, and writes the message to the standard input of the subshell. The command prints any error messages to standard output. This enables a wider ranging of paging customization.

The standard Zenoss Page Command is:

```
$ZENHOME/bin/zensnpp localhost 444 $RECIPIENT
```


This uses ZENHOME to send the page from the localhost; however, you can use any page command customizations you want. In this case \$RECIPIENT is actually the paging address for the user, as set in the settings for each user.

14.2. Event Manager Settings

You can adjust settings for the Event Manager, including:

- MySQL event database connection
- Event cache timeouts and counts
- Maintenance settings

14.2.1. Accessing Event Manager Settings

To access event manager settings, select Event Manager from the Navigation menu. The Event Manager Settings tab appears.

The screenshot shows the ZenEventManager interface with the 'Edit' tab selected. The left sidebar contains a navigation menu with sections: Main Views (Dashboard, Event Console, Device List, Network Map), Classes (Events, Devices, Services, Processes, Products), Browse By (Systems, Groups, Locations, Networks, Reports), and Management (Add Device, Mibs, Collectors, Settings, Event Manager). The main content area is titled '/ZenEventManager' and has tabs for Edit, Fields, History Fields, Commands, and Modifications. The 'Edit' tab is active and contains three sections: Connection Information, Cache, and Maintenance.

| Connection Information | |
|------------------------|--------------------|
| Backend Type | mysql |
| User Name | zenoss |
| Password | ***** |
| Database | events |
| Hostname | zenosst.zenoss.loc |
| Port | 3306 |

| Cache | |
|---------------------------|-----|
| Cache Timeout | 20 |
| Cache Clear Count | 20 |
| History Cache Timeout | 300 |
| History Cache Clear Count | 20 |

| Maintenance | |
|--|-------|
| Event Aging Threshold (hours) | 4 |
| Don't Age This Severity and Above | Error |
| Delete Historical Events Older Than (days) | 7 |
| Default Availability Report (days) | 7 |
| Default Syslog Priority | 3 |

At the bottom of the form, there are two buttons: 'Save Changes' and 'Save'.

Figure 14.2. Event Manager - Edit Tab

14.2.2. Changing Event Database Connection Information

To edit event database connection settings, make changes to one or more fields in the Connection area:

- **Backend Type** - Specifies the database type (MySQL). You cannot edit this value.
- **User Name** - Enter a user name for the MySQL database.
- **Password** - Enter the password for User Name.
- **Database** - Specify the database to use.
- **Hostname** - Specify the IP address of the host.

- **Port** - Specify the port to use when accessing the event database.

14.2.3. Changing Event Manager Cache Settings

To edit cache settings, make changes to one or more fields in the Cache area:

- **Cache Timeout** - Specify the cache timeout value for the event monitor. The lower the setting, the more responsive the event console will be.
- **Clear Cache Count** - Set the event count value at which the cache will be cleared of stored events.
- **History Cache Timeout** - Sets the timeout value for the History cache. The lower the number, the more responsible the history will be.
- **History Cache Clear Count** - Set the value at which history counts will be cleared.

14.2.4. Changing Event Manager Maintenance Settings

To edit maintenance settings, make changes to one or more fields in the Maintenance area:

- **Event Aging Threshold** (hours) - Specify how long Zenoss should wait before aging an event into the history table.
- **Don't Age This Severity and Above** - Select a severity level (Clear, Debug, Info, Warning, Error, or Critical). Events with this severity level and severity levels above this one will not age out and be placed into event history. (These events remain in the event list until acknowledged or moved into history manually.)
- **Delete Historical Events Older Than** (days) - Enter a value in days. Zenoss will automatically purge (delete) events from the event history that are older than this value.
- **Default Availability Report** (days) - Enter the number of days to include in the automatically generated Availability Report. This report shows a graphical summary of Zenoss availability and status.
- **Default Syslog Priority** - Specify the default severity level for an event to generate an entry into the syslog.

14.3. Setting Portlet Permissions

By setting portlet permissions, you determine which users can view and interact with Zenoss portlets. Permissions settings restrict which Zope Access Control List (ACL) can access each portlet.

Before you can successfully set portlet permissions, you must assign the user a specific Zenoss role. (You assign roles from the Users tab of the Settings area.) Each Zenoss user role is mapped to one or more Zope ACL permissions, which allow you to restrict the portlets a permission level can see.

A user's specific portlet permissions are defined in part by Zope ACL permissions, and in part by the role to which he is assigned.

14.3.1. User Role to ACL Mapping

The following table shows how user roles map to ACLs.

| User Roles | ACL Permission |
|----------------------------|-----------------------------|
| ZenUser, ZenOperator | ZenCommon, View |
| ZenManager, Manager | ZenCommon, View, Manage DMD |
| No Role, Administered Objs | ZenCommon |

14.3.2. Setting Permissions

To set portlet permissions:

1. From the Navigation menu, select Settings.
2. From Settings area, click **Portlets**.

The Portlets page appears.

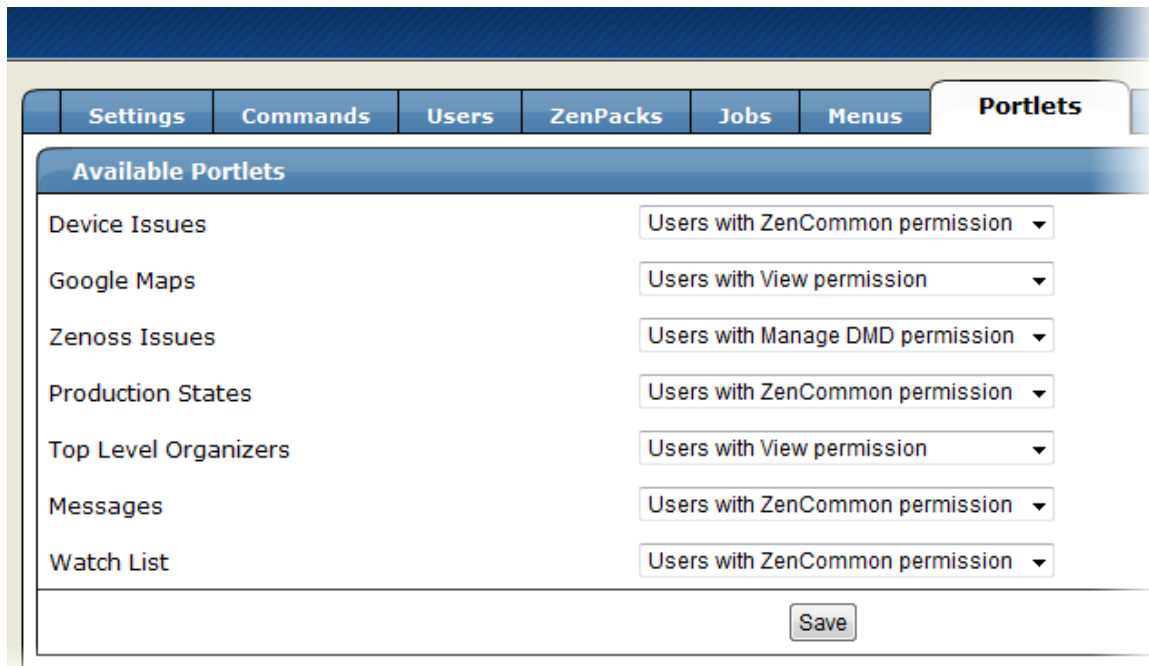


Figure 14.3. Portlet Permissions

3. For one or more portlets in the Available Portlets list, select the permission you want to apply.
4. Click **Save**.

14.3.3. Troubleshooting: Users Cannot See All Portlets

You may mistakenly block users from being able to access some portlets. Often, this happens when a user has been set to see only particular devices. By default, this user will see only portlets set to the ZenCommon permission level. In effect, this blocks three of six portlets.

To remedy this problem, you can:

- Change the permission levels (on the Portlets tab) to ZenCommon, or
- Change the user role to a role higher than "No Role."

14.4. Backup and Recovery

In some situations, you might want to back up configuration information and data from a Zenoss instance, and then later restore that instance. You might do this periodically, to take regular "snapshots" of your instance to archive; or infrequently, such as to move data from one Zenoss instance to another, or to restore a setup after performing a fresh installation of Zenoss. Zenoss provides tools that enable you to manage these backup and restore tasks.

With backup and restore, Zenoss includes:

- Events database (in MySQL)
- Zope database, which includes all devices, users, and event mappings
- `$ZENHOME/etc` directory, which contains configuration files for the Zenoss daemons
- `$ZENHOME/perf` directory, which contains performance data

Suggestions for a successful backup and restore experience:

- If you have the available disk space, tar and zip `$ZENHOME` before starting any backup or restore operation.
- Make sure Zenoss, including all daemons, is stopped before performing a restore operation.
- Avoid using these tools to go from a newer version of Zenoss to an older version.
- If you use these tools to go from an older version of Zenoss to a newer version, you should run **zenmigrate** after the restore operation.
- If restoring to a different Zenoss installation (one that differs from the backup version), make sure file paths in the `$ZENHOME/etc/*.conf` files are appropriate for the new environment after you restore.

The following sections describe Zenoss backup and restore scripts, as well as options for controlling their behavior.

14.4.1. Backup (zenbackup)


The Zenoss backup script is `$ZENHOME/bin/zenbackup`. Typical use of **zenbackup** looks like:

```
> zenbackup --save-mysql-access --file=BACKUPFILEPATH
```

If Zenoss is running then you can run **zenbackup** without any arguments, and a backup file will be placed in `$ZENHOME/backups`.

14.4.1.1. Backup Options

The following table lists frequently used **zenbackup** options.

 Use the **zenbackup --help** command to see a complete list of **zenbackup** options.

| Option | Description |
|----------------------------|---|
| --dbname | Specifies the name of the MySQL database Zenoss uses to hold event data. By default this is "Zenoss" but this can be specified when Zenoss is installed. This value can be seen by looking at the database field on the Event Manager page in Zenoss. If you do not specify --dbname then zenbackup will attempt to retrieve this information from ZEO unless you specify --dont-fetch-args . |
| --dbuser, --dbpass | These are the MySQL username/password used to access the events database. If you do not specify --dbuser or --dbpass then zenbackup will attempt to retrieve this information from ZEO unless you specify --dont-fetch-args . |
| --dont-fetch-args | This instructs zenbackup not to attempt to get values for <code>dbname</code> , <code>dbuser</code> and <code>dbpass</code> from ZEO. |
| --file=Filename | Use --file to specify a location for the backup file. By default it will be named <code>zenoss_Date.tgz</code> and placed in <code>\$ZENHOME/backups</code> . |
| --stdout | This flag tells zenbackup to send the backup information to stdout instead of to a file. Incompatible with --verbose . |
| --save-mysql-access | This instructs zenbackup to save <code>dbname</code> , <code>dbuser</code> and <code>dbpass</code> as part of the backup file so that zenrestore will have this information during a restore operation. Use this with caution as it means your backup files will contain a MySQL user name and password. |
| --no-eventsdb | Do not include the MySQL events database as part of the backup. |

| Option | Description |
|----------------------|--|
| -v, --verbose | Print progress messages. Incompatible with --stdout . |

14.4.1.2. Backups Tab

Zenoss provides a simple Web interface for creating and managing backups. Navigate to Settings > Backups to view the Backups page. The Create New Backup section allows you to create a backup through the GUI. The options available are a subset of those available with the **zenbackup** command line tool. Below that is the Backups section which lists all backup files in `$ZENHOME/backups`. You can delete one or more backup files by selecting them, and then selecting Delete Backup from the menu. Backup files can become large as your databases grow, so you may want to limit the number of backups you keep if drive space becomes an issue.

14.4.1.3. Remote Backups

Keeping backups on your Zenoss server should help you recover if one of your databases becomes corrupt or your configuration becomes problematic. However, Zenoss recommends that you keep at least one recent backup file on a different server (ideally at a different physical location) in case a physical disk fails.

14.4.2. Restore (zenrestore)

The Zenoss restore script is `$ZENHOME/bin/zenrestore`. Typical use of **zenrestore** looks like:

```
> zenrestore --file=BACKUPFILEPATH
```

14.4.2.1. Before You Restore (for Versions Earlier Than 2.4.5)


If you are running a version of Zenoss prior to 2.4.5, before you can restore your Zenoss instance, you must ensure that the same ZenPacks that were installed on the backup system are also installed on the target system.

Make sure that Zenoss is stopped before performing a restore.

If you used the **--save-mysql-access** option when you created the backup file then you only need to specify **--file** when you run **zenrestore**. Otherwise, you need to specify **dbname**, **dbuser** and **dbpass** also.

14.4.2.2. Restore Options

The following table lists frequently used **zenrestore** options.

 Use the **zenrestore --help** command to see a complete list of **zenrestore** options.

| Option | Description |
|---------------------------|---|
| --file | This is a backup file created with zenbackup . You must specify either --file or --dir . |
| --dir | The path to an unzipped backup file. You must specify either --file or --dir . |
| --dbname | This is the name of the MySQL database Zenoss uses to hold event data. This database must exist before zenrestore is run. If there are any Zenoss tables in the database they will be dropped by zenrestore before it restores the backed up tables and data. If you use a different dbname than was in use when the backup was created then after the restore, then you must set the database name on the Event Manager page. |
| --dbuser, --dbpass | These are the MySQL username/password used to access the events database. If you do not specify --dbuser or --dbpass then zenrestore will attempt to use values stored in the backup file if --save-mysql-access was used in creating it. |

| Option | Description |
|----------------------|--|
| --no-eventsdb | Do not restore the MySQL events database. If the backup file does not contain MySQL events data then zenrestore will not modify your events database even if you do not specify --no-eventsdb . |
| -v, --verbose | Print progress messages. |

14.5. Working with the Job Manager

The Zenoss Job Manager runs background tasks, such as discovering a network or adding a device. When you ask Zenoss to perform one of these tasks, it adds a job to the queue. Jobs are run by the `zenjobs` daemon.

i Not all jobs run in the Job Manager. When running other Zenoss jobs (in the foreground), do not navigate away from the current page until the job completes.

14.5.1. Viewing Jobs

To access the Zenoss Job Manager:

1. From the Navigation menu, click Settings.
2. From the Settings area, click the **Jobs** tab.

The jobs list appears.

| Zenoss server time: 12:59:53 | | | | | | |
|------------------------------|-----------------|-------------|----------------|-------------|-----------|--|
| Settings | Commands | Users | ZenPacks | Jobs | Menus | Portlets |
| Jobs | | | | | | |
| Status | Job Type | Description | Started | Finished | Duration | Actions |
| Running | ShellCommandJob | sleep 600 | 34 seconds ago | | | Click to stop the job. |
| Succeeded | ShellCommandJob | echo 1 | 3 hours ago | 3 hours ago | 0 seconds | Click to view the job log. Click to delete the job. |

Figure 14.4. Jobs List

The jobs list shows information about all jobs currently in the system:

- **Status** - Shows the current job status. Status options are Pending (waiting for `zenjobs` to begin running), Running, Succeeded, and Failed.
- **Job Type** - Provides a short indicator of the job type.
- **Description** - Provides a longer description of the job. Generally, this includes the shell command run by the `zenjobs` daemon.
- **Started / Finished / Duration** - Provide information about the time period in which the job ran.
- **Actions** - Shows actions you can take on the job. These include:
 - **Log** - Click to view the real-time output of a running job or final output from a completed job.
 - **Stop** - Ask the `zenjobs` daemon to stop running this job.
 - **Delete** - Remove this job from the system.

14.5.2. Running the zenjobs Daemon

You can stop and start the `zenjobs` daemon from the command line and the **Daemons** tab. You also can start it (if not already running) from the Job Manager. Click the link that appears in the jobs list.

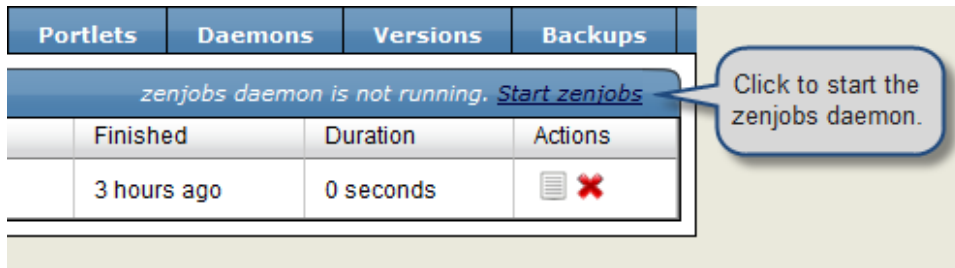


Figure 14.5. Start zenjob Daemon

14.6. Maintenance and Performance Tuning

Read the following sections for Zenoss maintenance and tuning suggestions.

14.6.1. Pack ZEO Database

The ZEO database needs to be packed periodically to reclaim space. To do this you should set up a cron job that runs the following command weekly:

```
$ZENHOME/bin/zeopack.py -p 8100
```

14.6.2. Log Rotate Script

The `logrotate` script must be present and running to rotate the Zope log (`event.log`), ZEO log (`zeo.log`), and access log (`z2.log`).

14.6.2.1. Zenoss 2.4.x

At installation, Zenoss adds the following script to the `/etc/logrotate.d/zenoss` file (where `ZENHOME` varies depending on your installation platform):

```
**ZENHOME**/log/event.log **ZENHOME**/log/z2.log **ZENHOME**/log/zeo.log{
    missingok
    weekly
    rotate 2
    copytruncate
}
```

In version 2.4.x, each Zenoss daemon performs its own log rotation. You can customize rotation parameters by inserting the following directives in each daemon's configuration file, located in the `$ZENHOME/etc` file:

| Directive | Description |
|--|--|
| <code>--maxlogsize=MAXLOGKILOBYTES</code> | Specifies the maximum size of the log file, in kilobytes. By default, the maximum is 10240 KB. |
| <code>--maxbackuplogs=MAXBACKUPLOGS</code> | Specifies the maximum number of backup log files. By default, the maximum is 3. |

If you do not specify a directive, then Zenoss uses the default values.

14.6.2.2. Zenoss 2.3.3 and Earlier

The `logrotate` script should be present in your installation, in the `/etc/logrotate.d/zenoss` file. For example, for a CentOS/RHEL, RPM-based installation, the following script should be present:

```
/opt/zenoss/log/*.log /opt/zenoss/log/*/*.log {
    missingok
    weekly
    rotate 2
    copytruncate
}
```

}

Appendix A. Zenoss Daemon Commands and Options

Zenoss daemons provide the services that collect and feed data to the Zenoss data layer. These daemons are divided among the following categories:

- Automated modeling
- Availability monitoring
- Event collection
- Performance monitoring
- Automated response

A.1. Automated Modeling Daemons

| Daemon | Description |
|------------|--|
| zendisc | Discovers new network resources. zendisc is a subclass of zenmodeler. It walks the routing table to discover the network topology, and then pings all discovered networks to find active IP addresses and devices. |
| zenmodeler | Configuration collection and configuration daemon, used for high-performance, automated model population. It uses SNMP, SSH, Telnet, and WMI to collect its information. zenmodeler works against devices that have been loaded into the DMD. It models devices on a periodic schedule (typically every 12 hours). |

A.2. Availability Monitoring Daemons

| Daemon | Description |
|------------|---|
| zenping | Performs high-performance, asynchronous testing of ICMP status. Uses the Zenoss Standard model to perform Layer 3 -aware network topology monitoring. |
| zenstatus | Performs active TCP connection testing of remote daemons. |
| zenprocess | Checks for the existence of monitored processes by using SNMP host resources' MIB, logging their CPU and memory utilization. For cases in which more than one process is matched, sums the CPU and memory and tracks the process count. |

A.3. Event Collection Daemons

| Daemon | Description |
|-------------|---|
| zensyslog | Collects and classifies syslog events. Parses the raw format to find the level and facility, host name, and tag (the freeform message string of the event). Syslog events often have specific, proprietary formats used by vendors; zensyslog tries to parse these by using a series of regular expressions defined in it. Once parsing is complete, the event is sent back to the event system (through zenhub) to be integrated with the model. |
| zeneventlog | Collects Windows Management Instrumentation (WMI) event log events. Forwards these events to zenhub for further processing. |
| zentrap | Collects SNMP traps, parses them, resolves OIDs into MIB names, and then forwards them to zenhub for further rules processing. |

A.4. Performance Monitoring Daemons

| Daemon | Description |
|-------------|--|
| zenperfsnmp | Performs high-performance, asynchronous SNMP performance collection and stores it locally to the collector. Thresholds are tested each time a value is written to disk. |
| zencommand | Performs XML RPC collection. Allows running of Nagios© and Cactii plug-ins on the local box, or on remote boxes through SSH. Commands must conform to the Nagios or Cactii API specifications. |

A.5. Automated Response Daemons

| Daemon | Description |
|------------|---|
| zenactions | Runs background jobs, such as email notification, database aging and maintenance window processing. |

Appendix B. SNMP Device Preparation

This appendix provide information about SNMP support, and lists Net-SNMP configuration settings that are required by Zenoss.

Add these lines to your `snmp.conf` file.

B.1. Net-SNMP

By default, Net-SNMP does not publish the full SNMP tree. Check to see if that is currently the case on a device and configure it correctly.

1. Confirm `snmpd` is running:

```
> snmpwalk -v1 -cpublic <your device name> system
```

2. Retrieve the IP table for the device with `snmpwalk`:

```
> snmpwalk -v1 -cpublic <your device name> ip
```

Typical SNMP View:

```
view systemview included .1
view systemview included .1.3.6.1.2.1.25.1
access notConfigGroup "" any noauth exact systemview none none
```

B.2. SNMP V3 Support

Zenoss provides support for SNMPv3 data collection.

The following `zProperties` control the authentication and privacy of these requests:

- `zSnmpAuthType`: use either "MD5" or "SHA" signatures to authenticate SNMP requests
- `zSnmpAuthPassword`: the shared private key used for authentication. Must be at least 8 characters long.
- `zSnmpPrivType`: either "DES" or "AES" cryptographic algorithms.
- `zSnmpPrivKey`: the shared private key used for encrypting SNMP requests. Must be at least 8 characters long.
- `zSnmpSecurityName`: the Security Name (user) to use when making SNMPv3 requests.

If `zSnmpPrivType` and `zSnmpPrivPassword` are set, the message is sent with privacy and authentication. If only the `zSnmpAuthType` and `zSnmpAuthPassword` are set, then the message is sent with Authentication but no Privacy. If neither the `Priv` or `Auth` values are set, the message is sent with no authentication or privacy. It is an error to set the `PrivType` and `PrivPassword` without also setting an `AuthType` and `AuthPassword`.

SNMPv3 encryption using the AES (Advanced Encryption Standard) algorithm is supported only if the host platform `net-snmp` library supports it.

Currently, RedHat 5 and Ubuntu 7.10 do not support AES, but OpenSuSE 10.2 and the Zenoss Appliance do.

You can determine if your platform supports AES by using the following test:

```
$ snmpwalk -x AES 2>&1 | head -1
```

If the response is:


```
"Invalid privacy protocol specified after -x flag: AES"
```

then your platform does not support AES encryption for SNMPv3.

If the response is:

```
"No hostname specified."
```

Then your platform supports AES.

 SNMPv3 Traps are not supported by Zenoss.

B.3. Community Information

This line will map the community name "public" into a "security name":

```
# sec.name source community
com2sec notConfigUser default public
```

This line will map the security name into a group name:

```
# groupName securityModel securityName
group notConfigGroup v1 notConfigUser
group notConfigGroup v2c notConfigUser
```

This line will create a view for you to let the group have rights to:

```
# Make at least snmpwalk -v 1 localhost -c public system fast again.
# name incl/excl subtree mask(optional)
view systemview included .1
```

This line will grant the group read-only access to the systemview view.

```
# group context sec.model sec.level prefix read write
notif
access notConfigGroup "" any noauth exact systemview
none none
```

B.4. System Contact Information

It is also possible to set the `sysContact` and `sysLocation` system variables through the `snmpd.conf` file:

```
syslocation Unknown (edit /etc/snmp/snmpd.conf)
syscontact Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
# Added for support of bcm5820 cards. pass .1 /usr/bin/ucd5820stat
```

B.5. Extra Information

For more information, see the `snmpd.conf` manual page, and the output of the `snmpd -H` command.

```
trapcommunity public
trapsink default
```

Appendix C. Using an Existing MySQL Server to Store Events

C.1. About

You can configure Zenoss to store events in an existing, remote MySQL server. You might want to do this if you think you may generate too many events for a local MySQL server to handle.

C.2. Procedure

The following steps show how an existing Zenoss installation can be configured to use a specific MySQL server.

1. Initialize the new database.


As a super-privileged user on the MySQL server, create the Zenoss events database schema. The `zenevents.sql` and `zenprocs.sql` files are located in `$ZENHOME/Products/ZenEvents/db` on your Zenoss server. Replace 10.1.2.30 with the IP address of your Zenoss server, and replace the password with your password.

From the MySQL client, run these commands:

```
CREATE DATABASE events;
\. zenevents.sql
\. zenprocs.sql
GRANT ALL PRIVILEGES ON events.* to zenoss@10.1.2.30 identified by 'password';
FLUSH PRIVILEGES;
```

2. Configure Zenoss to use the new database.

- a. In the Zenoss Web interface, navigate to Event Manager. (You must be assigned the Manager role to do this.)
- b. In the Connection Information section, adjust field values as needed. Replace 10.1.2.40 with the IP address of your MySQL server.
 - User Name = zenoss
 - Password = password
 - Database = events
 - Hostname = 10.1.2.40
 - Port = 3306
- c. Click **Save**.
- d. Restart Zenoss.

 The Zenoss installation `--help` option lists command-line options for setting up a remote MySQL server.

Appendix D. Syslog Device Preparation

D.1. Forwarding Syslog Messages from UNIX/Linux Devices

Zenoss has its own syslog server, zensyslog. Managed devices should point their syslog daemons to Zenoss. To do this, edit the `/etc/syslog.conf` file and add an entry, where 1.2.3.4 is the zensyslog IP:

1. Log on to the target device (as a super user).
2. Open `/etc/syslog.conf` file with a text editor (such as vi).
3. Enter `*.debug` and press the Tab key. Then enter the host name or IP address of the Zenoss server. See example below:

```
*.debug @192.168.X.X
```

4. Save the file and exit the file editor program.
5. Restart the Syslog service using the command below:

```
/etc/init.d/syslog restart
```

D.2. Forwarding Syslog Messages from a Cisco IOS Router

Here are some links to Cisco commands to turn on syslog. Typically, it is easier to use syslog than SNMP traps from network devices. The most basic IOS command to send syslog messages is:

```
logging 1.2.3.4
```

D.2.1. Other Cisco Syslog Configurations

Here are some additional configurations for other Cisco devices. To set up these configurations follow the following steps using the configurations that follow below.

1. Log on to the target router.
2. Type the command `enable` at the prompt.
3. Once you are prompted for a password, enter the correct password.
4. Type the command `config` at the prompt.
5. Type the command `terminal` at the configuration prompt.
6. At the prompt, Set the Syslog forwarding mechanism. See example below:

```
logging <IP address of the Zenoss server>
```

7. Exit out all the prompts to the main router prompt.

Catalyst

```
set logging server enable
set logging server 192.168.1.100
set logging level all 5
set logging server severity 6
```

Local Director

```
syslog output 20.5
no syslog console
syslog host 192.168.1.100
```

PIX Firewalls

```
logging on
logging standby
```

```
logging timestamp
logging trap notifications
logging facility 19
logging host inside 192.168.1.100
```

D.3. Forwarding Syslog Messages from a Cisco CatOS Switch

1. Log on to the target switch.
2. Type the command enable at the prompt.
3. Once you are prompted for a password, enter the correct password.
4. Set the Syslog forwarding mechanism. See example below:

```
set logging server <IP address of the Zenoss server>
```

5. You can set the types of logging information that you want the switch to provide with the commands below as examples:

```
set logging level mgmt 7 default
set logging level sys 7 default
set logging level filesys 7 default
```

D.4. Forwarding Syslog Messages using Syslog-ng

Here is an example for FreeBSD and Linux platforms.

1. Log on to the target device (as a super user)
2. Open /etc/syslog-ng/syslog-ng.conf file with a text editor (e.g VI).
3. Add source information to file. See example below:

FreeBSD:

```
source src { unix-dgram("/var/run/log"); internal ();};
```

Linux: (will gather both system and kernel logs)

```
source src {
internal();
unix-stream("/dev/log" keep-alive(yes) max-connections(100));
pipe("/proc/kmsg");
udp();
};
```

4. Add destination information (in this case, the Zenoss server). See example below:

```
log { source(src); destination(zenoss); };
```

Appendix E. TALES Expressions

E.1. About Tales Expressions

Use TALES syntax to retrieve values and call methods on Zenoss objects. Several fields in Zenoss accept TALES syntax; these include:

- Command templates
- User commands
- Event commands
- zLinks

Commands (those associated with devices and those associated with events) can use TALES expressions to incorporate data from the related devices or events. TALES is a syntax for specifying expressions that let you access the attributes of certain objects, such as a device or an event in Zenoss.

For additional documentation on TALES syntax, see the TALES section of the Zope Page Templates Reference:

http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/AppendixC.stx

Depending on context, you may have access to a device, an event, or both. Following is a list of the attributes and methods you may want to use on device and event objects. The syntax for accessing device attributes and methods is `${dev/attributename}`. For example, to get the `manageIp` of a device you would use `${dev/manageIp}`. For events, the syntax is `${evt/attributename}`.

A command to ping a device might look like this. (The `${..}` is a TALES expression to get the `manageIp` value for the device.)

```
ping -c 10 ${device/manageIp}
```

E.1.1. Examples

- DNS Forward Lookup (assumes device/id is a resolvable name)

```
host ${device/id}
```

- DNS Reverse Lookup

```
host ${device/manageIp}
```

- SNMP Walk

```
snmpwalk -v1 -c${device/zSnmpCommunity} ${device/manageIp} system
```

To use these expressions effectively, you must know which objects, attributes, and methods are available, and in which contexts. Usually there is a device that allows you to access the device in a particular context. Contexts related to a particular event usually have event defined.

E.2. TALES Device Attributes

The following table lists available device attributes.

| Attribute | Description |
|-----------------|--|
| getId | The primary means of identifying a device within Zenoss |
| getManageIp | The IP address used to contact the device in most situations |
| productionState | The production status of the device: Production, Pre-Production, Test, Maintenance or Decommissioned. This attribute is a numeric value, use <code>getProductionStateString</code> for a textual representation. |

| Attribute | Description |
|-----------------------------|--|
| getProductionStateString | Returns a textual representation of the productionState |
| snmpAgent | The agent returned from SNMP collection |
| snmpDescr | The description returned by the SNMP agent |
| snmpOid | The oid returned by the SNMP agent |
| snmpContact | The contact returned by the SNMP agent |
| snmpSysName | The system name returned by the SNMP agent |
| snmpLocation | The location returned by the SNMP agent |
| snmpLastCollection | When SNMP collection was last performed on the device. This is a DateTime object. |
| getSnmpLastCollectionString | Textual representation of snmpLastCollection |
| rackSlot | The slot name/number in the rack where this physical device is installed |
| comments | User entered comments regarding the device |
| priority | A numeric value: 0 (Trivial), 1 (Lowest), 2 (Low), 3 (Normal), 4 (High), 5 (Highest) |
| getPriorityString | A textual representation of the priority |
| getHWManufacturerName | Name of the manufacturer of this hardware |
| getHWProductName | Name of this physical product |
| getHWProductKey | Used to associate this device with a hardware product class |
| getOSManufacturerName | Name of the manufacturer of this device's operating system. |
| getOSProductName | Name of the operating system running on this device. |
| getOSProductKey | Used to associate the operating system with a software product class |
| getHWSerialNumber | Serial number for this physical device |
| getLocationName | Name of the Location assigned to this device |
| getLocationLink | Link to the Zenoss page for the assigned Location |
| getSystemNames | A list of names of the Systems this device is associated with |
| getDeviceGroupNames | A list of names of the Groups this device is associated with |
| getOsVersion | Version of the operating system running on this device |
| getLastChangeString | When the last change was made to this device |
| getLastPollSnmpUpTime | Uptime returned from SNMP |
| uptimeStr | Textual representation of the SNMP uptime for this device |
| getPingStatusString | Textual representation of the ping status of the device |
| getSnmpStatusString | Textual representation of the SNMP status of the device |

E.3. Tales Event Attributes

The following table lists available event attributes.

| Attribute | Description |
|-----------|---|
| agent | Collector name from which the event came (such as zensyslog or zentrap). |
| component | Component of the associated device, if applicable. (Examples: eth0, httpd.) |
| count | Number of times this event has been seen. |
| dedupid | Key used to correlate duplicate events. By default, this is: device, component, eventClass, eventKey, severity. |

| Attribute | Description |
|---------------|--|
| device | ID of the associated device, if applicable. |
| DeviceClass | Device class from device context. |
| DeviceGroups | Device systems from device context, separated by . |
| eventClass | Event class associated with this device. If not specified, may be added by the rule process. If this fails, then will be /Unknown. |
| eventClassKey | Key by which rules processing begins. Often equal to component. |
| eventGroup | Logical group of event source (such as syslog, ping, or nteventlog). |
| eventKey | Primary criteria for mapping events into event classes. Use if a component needs further de-duplication specification. |
| eventState | State of event. 0 = new, 1 = acknowledged, 2 = suppressed. |
| evid | Unique ID for the event. |
| facility | syslog facility, if this is a syslog event. |
| firstTime | UNIX timestamp when event is received. |
| ipAddress | IP Address of the associated device, if applicable. |
| lastTime | Last time this event was seen and its count incremented. |
| Location | Device location from device context. |
| manager | Fully qualified domain name of the collector from which this event came. |
| message | Full message text. |
| nteid | nt event ID, if this is an nt eventlog event. |
| priority | syslog priority, if this is a syslog event. |
| prodState | prodState of the device context. |
| severity | One of 0 (Clear), 1 (Debug), 2 (Info), 3 (Warning), 4 (Error) or 5 (Critical). |
| stateChange | Time the MySQLrecord for this event was last modified. |
| summary | Text description of the event. Limited to 150 characters. |
| suppid | ID of the event that suppressed this event. |
| Systems | Device systems from device context, separated by . |

zProperties and Custom Properties

zProperties and custom properties also are available for devices, and use the same syntax as shown in the previous sections.

Glossary

| | |
|---------------------|--|
| alert | Email or page sent as a result of an event. |
| data point | Data returned from a data source. In many cases, there is only one data point for a data source (such as in SNMP); but there may also be many data points for a data source (such as when a command results in the output of several variables). |
| data source | Method used by Zenoss to collect monitoring information. Example data sources include SNMP OIDs, SSH commands, and perfmon paths. |
| device | Primary monitoring object in Zenoss. Generally, a device is the combination of hardware and an operating system. |
| device class | Special type of organizer used to manage how the system models and monitors devices (through zProperties and monitoring templates). |
| device component | Object contained by a device. Components include interfaces, OS processes, file systems, CPUs, and hard drives. |
| discovery | Process by which Zenoss gathers detailed information about devices in the infrastructure. Results of discovery are used to populate the Zenoss model. |
| event | Manifestation of important occurrence within the system. Events are generated internally (such as when a threshold is exceeded) or externally (such as through a syslog message or SNMP trap). |
| event class | Categorization system used to organize event rules. |
| event rules | Controls how events are manipulated as they enter the system (for example, changing the severity of an event). zProperties configure event rules. |
| graph | Displays one or more data points, thresholds, or both. |
| managed resource | Servers, networks, and other devices in the IT environment. |
| model | Representation of the IT infrastructure in Zenoss. The model tells Zenoss "what is out there" and how to monitor it. |
| monitoring template | Description of what to monitor on a device or device component. Monitoring templates comprise four main elements: data sources, data points, thresholds, and graphs. |
| organizer | Hierarchical system used to describe locations and groups. Zenoss also includes special organizers, which are classes that control system configuration. |
| resource component | Interfaces, services and processes, and installed software in the IT environment. |
| threshold | Defines a value beyond which a data point should not go. When a threshold is reached, Zenoss generates an event. Typically, threshold events use the / Perf event class. |
| zProperty | Configuration property defined on a device or event class. zProperties control a large part of how monitoring is performed in Zenoss. Configuration of zProperties relies on inheritance. |