# Creating Zenoss ZenPacks for Zenoss 3

***Version 2***

***DRAFT***

***Updated January 2011***

***Jane Curry***

***Skills 1st Ltd***

***[www.skills-1st.co.uk](http://www.skills-1st.co.uk)***

Jane Curry
Skills 1st Ltd
2 Cedar Chase
Taplow
Maidenhead
SL6 0EU
01628 782565

jane.curry@skills-1st.co.uk

## Synopsis

ZenPacks are the extension mechanism provided by Zenoss to build new functionality and also to easily port customisation from one Zenoss server to another. Some documentation is provided in the Zenoss Developer's Guide; this paper is intended to enhance and extend that documentation, including a sample ZenPack.

The process of creating, modifying and exporting ZenPacks is discussed, along with debugging hints. The sample ZenPack explores:

- creating new object classes and relationships
- creating new collector modeler plugins to populate the new classes with data
- creating skins to display web pages for the new types of object
- creating JavaScript to display components of devices
- creating performance data templates for the object classes

It is assumed that the reader is familiar with basic SNMP concepts and with standard Zenoss configuration techniques.

This paper was originally written based on a stack-built Zenoss Core 2.4.1 on SuSE 10.3. The updated paper is based on a stack-built Zenoss 3.0.3 on SuSE 10.3. All commands and menu options have been updated for Zenoss 3, unless otherwise stated.

The hostname of the Zenoss server in the updated paper is *zen3.class.example.org.*

## Notations

Throughout this paper, text to by typed, file names and menu options to be selected, are highlighted by *italics;* important points to take note of are shown in **bold.**

## Table of Contents

# 1 What are ZenPacks?

ZenPacks are the method of extending the standard Zenoss functionality. There are four different sources of ZenPacks:

- Zenoss Core ZenPacks that can be downloaded from http://www.zenoss.com/community/projects/zenpacks/ . These are developed and maintained by Zenoss and are available to both Zenoss Core and Zenoss Enterprise users. They include monitoring of Apache, Dell, FTP, HTTP, LDAP, JMX and MySQL, amongst others.
- Zenoss community ZenPacks, also from http://www.zenoss.com/community/projects/zenpacks/ . These are ZenPacks developed by individuals or organisations and made freely available to the Zenoss community. No support should be implied for them. There are a large number of community ZenPacks covering the monitoring of VMware, wireless devices, Cisco devices, various switches, printers and several ZenPacks to enhance the reporting of Zenoss devices, events and thresholding.
- Zenoss Enterprise ZenPacks are available at no extra charge to Zenoss Enterprise (ie. paying) customers. They include enhanced VMware and Windows monitoring, fine-grained user management, distributed monitoring and high availability, and a global dashboard, as well as enhanced monitoring of many third-party devices and software packages.
- Write your own ZenPack – and optionally make it available as a community ZenPack

Since Zenoss 2.2, ZenPacks are packaged as Python eggs. Earlier zip format ZenPacks can be converted to eggs (see the reference document at http://community.zenoss.org/docs/DOC-2372 ). This packaging is performed automatically for you and you don't need to get into the details of eggs.

Some of the core and community ZenPacks come with their own documentation; sometimes it is a little sparse. Searching the Zenoss forums is a good way to glean information ( http://community.zenoss.org/community/forums ).

ZenPacks may be used for two main reasons:

- Creating new monitoring of new types of devices
- Porting either standard or ZenPack configuration of Zenoss, from one Zenoss server to another

Many of the standard Zenoss Graphical User Interface (GUI) menus have an *Add to ZenPack* option; thus event classes, event commands, user commands, device classes, service classes, process classes, reports and product definitions as well as the the data sources, graphs and thresholds of performance templates, can be simply added to a ZenPack using the GUI ( a **simple** ZenPack).

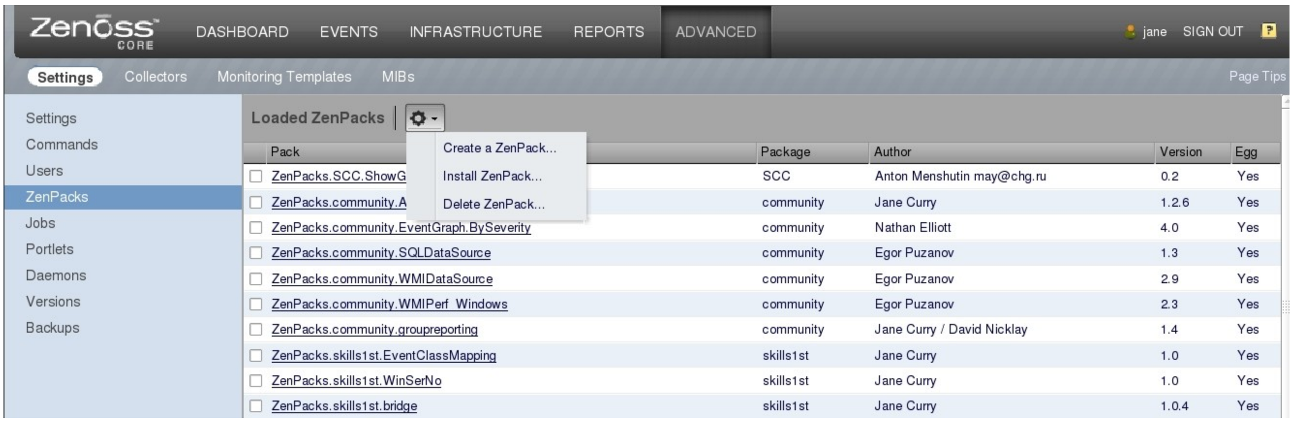
A ZenPack can also add daemons, new device types and user interface features such as menus but this requires programming effort ( a **complex** ZenPack). Check Chapter 13 of the Zenoss 3 Administration Guide for a short introduction to ZenPacks.

# 2 The process of building a ZenPack

Before diving into the complexities of writing Python code for complex ZenPacks, step back and examine the **process** that is required. The first question is whether this will be a simple ZenPack that can be entirely created from the GUI, or whether code needs to be written. Either way, the process for creating the ZenPack is exactly the same.

## 2.1 ZenPack creation

As a Zenoss user with the *Manager* role, use the top-level *ADVANCED -> Settings* option and select *ZenPacks* from the left-hand menu.



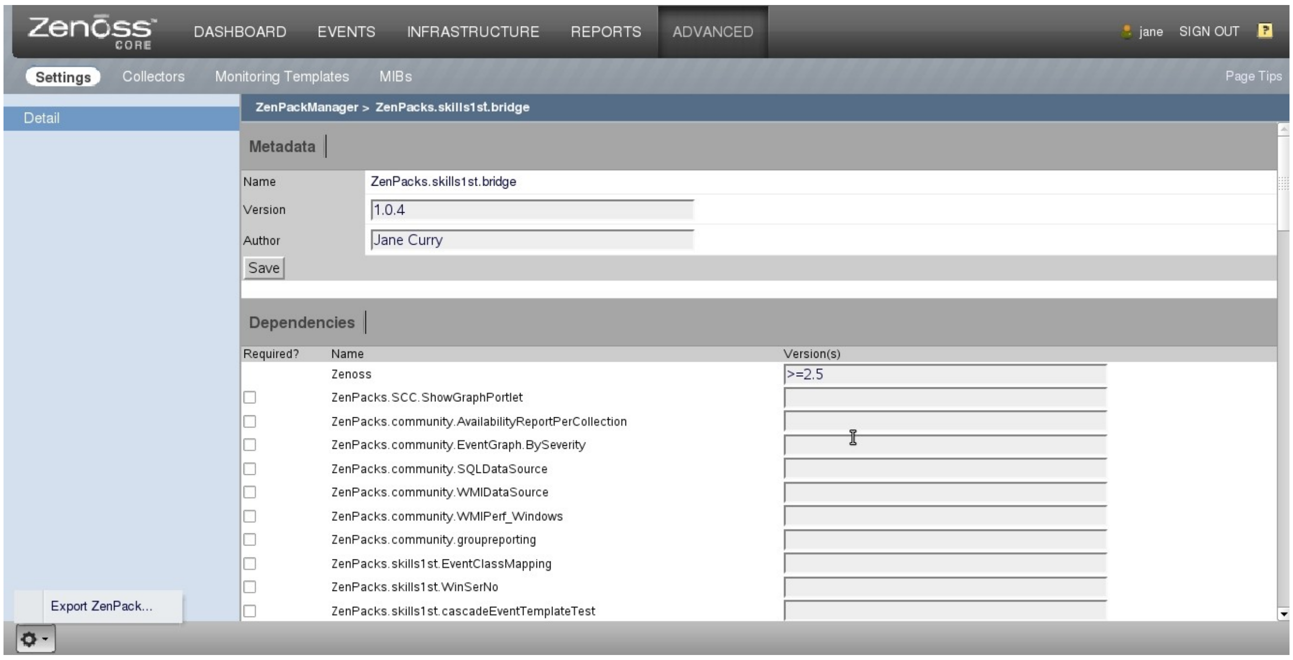
| Pack | Package | Author | Version | Egg |
|---|---|---|---|---|
| ZenPacks.SCC.ShowG | SCC | Anton Menshutin may@chg.ru | 0.2 | Yes |
| ZenPacks.community.A | community | Jane Curry | 1.2.6 | Yes |
| ZenPacks.community.EventGraph.BySeverity | community | Nathan Elliott | 4.0 | Yes |
| ZenPacks.community.SQLDataSource | community | Egor Puzanov | 1.3 | Yes |
| ZenPacks.community.WMIDataSource | community | Egor Puzanov | 2.9 | Yes |
| ZenPacks.community.WMIPerf_Windows | community | Egor Puzanov | 2.3 | Yes |
| ZenPacks.community.groupreporting | community | Jane Curry / David Nicklay | 1.4 | Yes |
| ZenPacks.skills1st.EventClassMapping | skills1st | Jane Curry | 1.0 | Yes |
| ZenPacks.skills1st.WinSerNo | skills1st | Jane Curry | 1.0 | Yes |
| ZenPacks.skills1st.bridge | skills1st | Jane Curry | 1.0.4 | Yes |

*Figure 1: ZenPacks option from the ADVANCED -> Settings menu*

The *Action* icon (the "gear" icon at the top of the main panel) then offers the following options:

- Create a ZenPack
- Install ZenPack
- Delete ZenPack

When creating a new ZenPack, the first thing you are asked for is the ZenPack name. ZenPack names are a sequence of three package names separated by periods. The first part of the name is always **ZenPacks**. The second part usually identifies the person or organization responsible for the ZenPack. The last part of the name usually identifies the function of the ZenPack (see the screenshot above for examples). Once named, you can then specify other parameters for your ZenPack, like Zenoss version dependency or other co-requisite ZenPacks. You should also specify an author and a version for this ZenPack.



ZenPackManager > ZenPacks.skills1st.bridge

Metadata

| | |
|---|---|
| Name | ZenPacks.skills1st.bridge |
| Version | 1.0.4 |
| Author | Jane Curry |

Save

Dependencies

| Required? | Name | Version(s) |
|---|---|---|
| | Zenoss | >=2.5 |
| ☐ | ZenPacks.SCC.ShowGraphPortlet | |
| ☐ | ZenPacks.community.AvailabilityReportPerCollection | |
| ☐ | ZenPacks.community.EventGraph.BySeverity | |
| ☐ | ZenPacks.community.SQLDataSource | |
| ☐ | ZenPacks.community.WMIDataSource | |
| ☐ | ZenPacks.community.WMIPerf_Windows | |
| ☐ | ZenPacks.community.groupreporting | |
| ☐ | ZenPacks.skills1st.EventClassMapping | |
| ☐ | ZenPacks.skills1st.WinSerNo | |
| ☐ | ZenPacks.skills1st.cascadeEventTemplateTest | |

*Figure 2: Creation details for a ZenPack*

When you create the ZenPack, a directory hierarchy is created under $ZENHOME/ZenPacks (note that older style, pre Zenoss 2.2 ZenPacks, used $ZENHOME/Products as the base directory). Each of the directories will have a largely-empty **__init__.py** file that needs to be there but you probably won't need to modify it.

The main directory areas that will be discussed in this paper, for the ZenPack called ZenPacks.skills1st.bridge, are:

- ZenPacks.skills1st.bridge – the base ZenPack directory. It contains object class definition files
- ZenPacks.skills1st.bridge/modeler/plugins - modeler plugins for object classes
- ZenPacks.skills1st.bridge/skins/ZenPacks.skills1st.bridge - contains skins files describing web page templates associated with displaying aspects of the new object classes

Some of these are rather long-winded but they are created automatically and that is what we have to go with! Once the structure is created, "things" can be added to the ZenPack either from the GUI using *Add to ZenPack* menu options (this is known as **development mode**), or programmatically by placing files in the appropriate directories ( **source mode**); indeed, both these methods can be used at any stage.

## 2.2 Exporting and installing ZenPacks

When you are ready to test the ZenPack on a different system it needs to be exported to create the Python egg file. Note that the export process also creates the *objects/objects.xml* file - more of this later. From the *Detail* page of the ZenPack, use the *Action* icon at the bottom of the left-hand menu to *Export ZenPack*.

The options presented are:

- Export to $ZENHOME/exports
- Export to $ZENHOME/exports and download

Typically you leave the top radio button selected to just create the ZenPack egg file in $ZENHOME/exports. The file is first created in your ZenPack's **dist** directory then copied to the $ZENHOME/exports directory.

The .egg file can now be moved to a different Zenoss server (perhaps you have a test and a production server?) and installed as any other ZenPack, either using the *ADVANCED -> Settings -> ZenPacks* menus and then the *Action* icon option to *Install ZenPack*; or you can use the command line:

```
zenpack --install ZenPacks.skills1st.bridge-1.0.4-py2.6.egg
```

Note the syntax here is 2 hyphens preceding the *install*.

The formal documentation varies somewhat as to what daemons you need to recycle after importing a ZenPack. *zenoss restart* would always be safe but bounces **all** of the daemons. I believe that *zenhub restart* and *zopectl restart* is sufficient. Note that if you forget to recycle the daemons, you may well get error messages from the ZenPacks page and from ZenPack functionality.

When you install an egg ZenPack, you usually don't have the ability to modify it, thought is possible to do so – see Chapter3, page 25 of the Zenoss 3 Developer's Guide for instructions.

If you wish to continue to develop the ZenPack on the new system, the other alternative is to copy the whole ZenPack directory structure and then install the ZenPack with a *--link* parameter (2 hyphens again). This is good practise during initial development as well as in the scenario where you wish to export a ZenPack and then continue to modify it on a different system, largely because if you accidentally use the *Remove ZenPack* menu, it deletes **all** files relating to that ZenPack under $ZENHOME/ZenPacks and this will include any development code you have created if it is stored there.

The sample ZenPack discussed in this paper was created as described above and then moved out of the $ZENHOME/ZenPacks directory using:

```
cp -r $ZENHOME/ZenPacks/ZenPacks.skills1st.bridge $ZENHOME/local
zenpack --link --install $ZENHOME/local/ZenPacks.skills1st.bridge
```

It is perfectly acceptable to reinstall a ZenPack that already exists – it will simply give a warning message that the ZenPack is already installed, but it will do the install. Remember to restart zenhub and zopectl.

The result of the --link parameter is to replace the *ZenPacks.skills1st.bridge* directory hierarchy in the standard $ZENHOME/ZenPacks directory with a single file, *ZenPacks.skills1st.bridge.egg-link*, which simply contains the base directory of where your ZenPack really is. Now, if anyone removes this ZenPack, the only thing that is deleted from $ZENHOME/ZenPacks is this link file, not all your ZenPack code.

From this point, you can continue to develop the ZenPack, either in Development mode, or by writing code in appropriate directories; a mixture of both is perfectly acceptable and all changes will follow this link to actually update code in your private directory.

Zenoss requires a Python module called *setuptools* to create and install eggs. In Zenoss 3, the setuptools module is under *$ZOPEHOME* (which for a SuSE install translates to */usr/local/zenoss/python*) and then down *lib/python2.6/site-packages* . Zenoss also provides a module named *zenpacksupport* which extends setuptools . The zenpacksupport class defines additional metadata that is written to and read from ZenPack eggs. This metadata is provided through additional options passed to the *setup()* call in a ZenPack's *setup.py* file.

When a ZenPack is exported, it automatically creates an egg file whose name includes the python version, where “2.4” represents Zenoss 2 and “2.6” represents Zenoss 3 (for example *ZenPacks.skills1st.bridge-1.0.4-py2.6.egg)* . Attempting to install a 2.6 egg file in a Zenoss 2 environment and vice versa, will often fail with a message including “***BLOCKED*** by –allow-hosts”. Simply renaming the egg file to have the correct 2.x in the filename often circumvents the problem.

If you want to work with ZenPacks from the Zenoss ZenPacks website, checking out and in existing ZenPacks, there is an excellent document by David Buler, “ZenPack Development Procedures” at http://community.zenoss.org/docs/DOC-10223 .

# 3 “Simple” ZenPacks

Some ZenPacks can simply be created using the Zenoss GUI; this is especially useful for moving standard configurations from one Zenoss server to another but may also be appropriate when creating ZenPacks to share with other people.

The ZenPack is created exactly as described in chapter 2 above. To add “things” to the ZenPack, simply use the *Add to ZenPack* option that is available on many of the dropdown menus. The following can be added from menus (ie. in development mode):

- Device Classes
- Event Classes
- Event Mappings
- User Commands
- Event Commands

- MIBs
- Service Classes
- Process Classes
- Device Organizers
- Performance Templates

You will be prompted as to which ZenPack you wish to add the item to. Objects can be removed from the ZenPack by selecting the checkboxes next to them and using the *Delete from ZenPack* menu item. Devices themselves 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 see what a ZenPack contains, simply use the *ZenPacks* option from the *ADVANCED -> Settings* menu and choose the appropriate ZenPack.

When a ZenPack is **exported** (using the *Action* icon from the Detail page of the ZenPack), not only is the egg file created but it is at this time that all the objects under the "ZenPack Provides" list, are written to the **objects.xml** file under the objects directory of the ZenPack. This file can be inspected with an editor – as the name suggests, it is in xml format.

# 4 Designing complex ZenPacks

When developing new functionality for Zenoss with ZenPacks, some tasks require more than the standard customisation tools can capture using development mode. For example:

- Supporting a different type of device with different attributes eg. a switch that supports the Bridge MIB
- Polling for SNMP variables from the Bridge MIB to populate these new attributes, such as Port number and the MAC address of the remote device connected to that port
- Displaying web pages that show information about the new device types and their attributes
- Creating new daemons to gather either configuration polling information (modeling) or performance data. Data collection methods for SNMP and ssh are provided as standard but you may need JMX, HTTP or any other method (there are Core ZenPacks that support JMX and HTTP).

This paper will examine the first three of these in detail.

## 4.1 Basic principles

Before discussing the sample ZenPack requirements and its implementation, let's get some basic principles straight first.

### 4.1.1 Configuration data and performance data

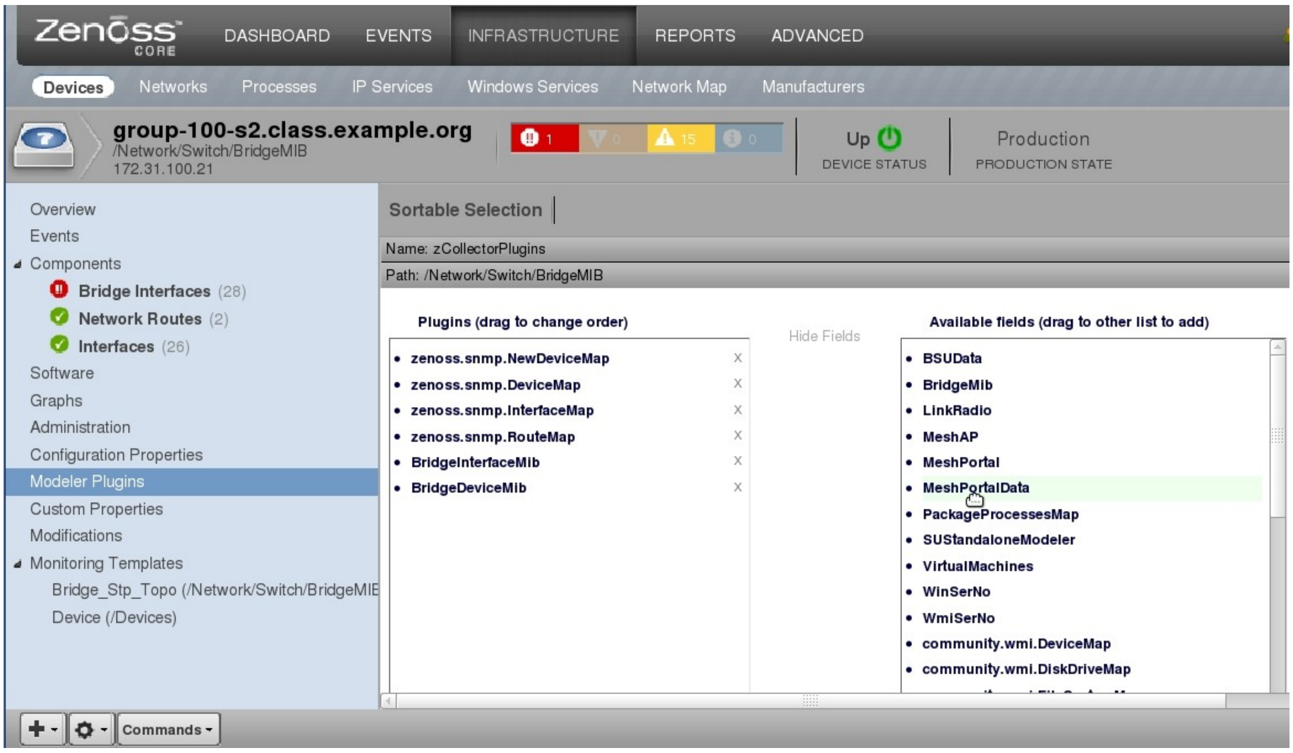
Zenoss documentation is apt to be a little imprecise sometimes in its terminology and to use different words to mean the same thing. There are two very different concepts to do with collecting data. **Configuration** data is typically polled for every 12 hours and is held in the Zope Object Database (ZODB). **Performance** data is typically polled for every 5 minutes and is held in Round Robin Database (RRD) files from where it can be graphed. The two are very different.

Configuration data is polled for by the **zenmodeler** daemon, using **modeler plugins,** also sometimes called **collector plugins**. Lots of these are provided as standard with Zenoss under $ZENHOME/Products/DataCollector/plugins/zenoss with separate subdirectories for:

- cmd
- nmap
- portscan
- python
- snmp

Don't be fooled by the directory path containing "DataCollector" - these are configuration modeler plugins used by the zenmodeler daemon and nothing to do with the collection of performance data that typically is collected by the zenperfsnmp or zencommand daemons.

Any device or device class can have several modeler plugins assigned to it. This is configured from the left-hand *Modeler Plugins* menu of a device's Detail page or, for a device class, follow the *DETAILS* link at the top of the left-hand menu for the equivalent *Modeler Plugins* option.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Up — DEVICE STATUS
Production — PRODUCTION STATE

Bridge Interfaces (28)
Network Routes (2)
Interfaces (26)

Bridge_Stp_Topo (/Network/Switch/BridgeMIB
Device (/Devices)

Sortable Selection

Name: zCollectorPlugins
Path: /Network/Switch/BridgeMIB

Plugins (drag to change order)

- zenoss.snmp.NewDeviceMap
- zenoss.snmp.DeviceMap
- zenoss.snmp.InterfaceMap
- zenoss.snmp.RouteMap
- BridgeInterfaceMib
- BridgeDeviceMib

Hide Fields

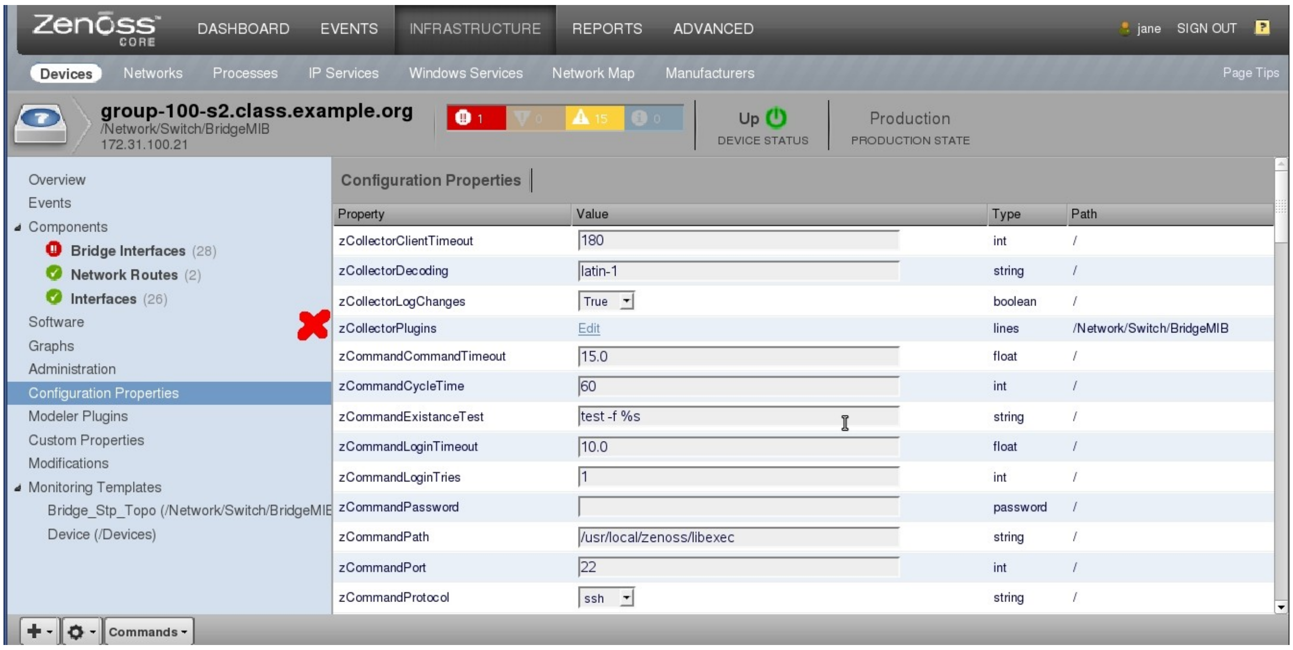
Available fields (drag to other list to add)

- BSUData
- BridgeMib
- LinkRadio
- MeshAP
- MeshPortal
- MeshPortalData
- PackageProcessesMap
- SUStandaloneModeler
- VirtualMachines
- WinSerNo
- WmiSerNo
- community.wmi.DeviceMap
- community.wmi.DiskDriveMap

*Figure 3: The Modeler Plugins dialogue for a specific device*

Initially the dialogue shows the modeler plugins that are currently assigned on the left, and the right of the dialogue has “Add Fields”, greyed out. Although the option appears to be greyed out, click it to see the other modeler plugins that exist. They can be selected simply by dragging them to the assigned area; the order the plugins are run can be changed by dragging the plugins to the appropriate order. Don't forget to use the *Save* button.

Another way to achieve exactly the same effect is to go to the device class or individual device's zProperties page and click on the *Edit* button beside *zCollectorPlugins*, which takes you to exactly the same dialogue shown in Figure 3.



Configuration Properties

| Property | Value | Type | Path |
|---|---|---|---|
| zCollectorClientTimeout | 180 | int | / |
| zCollectorDecoding | latin-1 | string | / |
| zCollectorLogChanges | True | boolean | / |
| zCollectorPlugins | Edit | lines | /Network/Switch/BridgeMIB |
| zCommandCommandTimeout | 15.0 | float | / |
| zCommandCycleTime | 60 | int | / |
| zCommandExistanceTest | test -f %s | string | / |
| zCommandLoginTimeout | 10.0 | float | / |
| zCommandLoginTries | 1 | int | / |
| zCommandPassword | | password | / |
| zCommandPath | /usr/local/zenoss/libexec | string | / |
| zCommandPort | 22 | int | / |
| zCommandProtocol | ssh | string | / |

*Figure 4: Modify the zCollectorPlugins zProperty to activate modeler plugins*

As is usual with Zenoss, modeler plugins should be assigned as high as possible in the device class hierarchy to prevent unnecessary configuration and all sub device classes and devices will inherit that property; modeler plugins can always be deconfigured for a specific device if necessary.

There is no need to use the *Action* icon and *Push Changes* menu; new modeler plugins will be automatically applied the next time zenmodeler is run or a manual remodel is performed (*Action* icon -> *Model Device*).

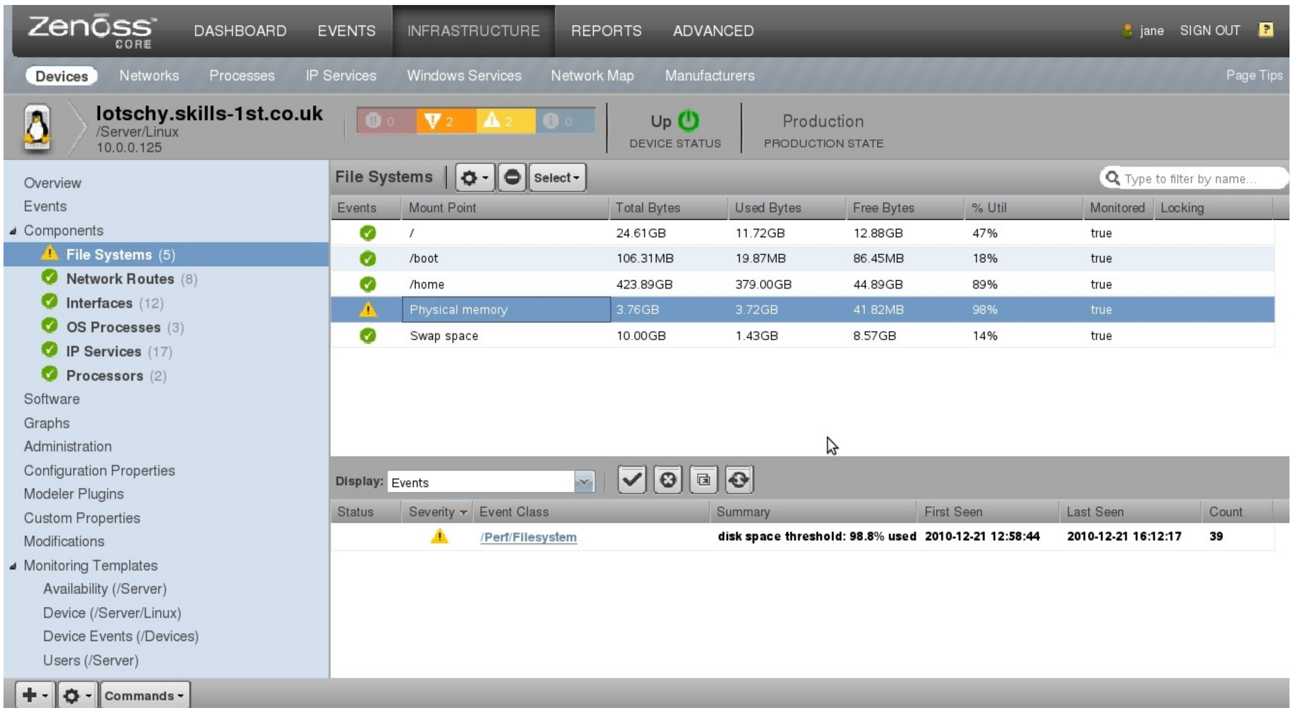
Alternatively, for a specific device called switch.skills-1st.co.uk, use the following command line. The *-v 10* turns on debugging to loglevel 10 (the highest level).

```
	zenmodeler run -v 10 -d switch.skills-1st.co.uk
```

You should see each of the modeler plugins listed and some results from each plugin.

Performance data to be collected is specified using Zenoss **templates**. As with modeler plugins, templates can be assigned either to a device class hierarchy or to a specific device but the definition of these templates, the RRD databases that contain the data and the daemons that collect the data are entirely separate from the configuration data collection mechanism. If you can access performance data using either SNMP or ssh then, typically, there is no need to write new code to collect performance data.

Modeler plugins are run by the zenmodeler daemon whereas SNMP performance template data is collected by **zenperfsnmp** and ssh-driven performance data is collected by the **zencommand** daemon. Another significant difference between modeler plugins and performance templates is that if thresholds set in performance templates are exceeded, or if performance data collection fails, then indications will show for the Component on the device's Detail page, as shown in Figure 5.



lotschy.skills-1st.co.uk
/Server/Linux
10.0.0.125

0 | 2 | 2 | 0

Up
DEVICE STATUS

Production
PRODUCTION STATE

File Systems

| Events | Mount Point | Total Bytes | Used Bytes | Free Bytes | % Util | Monitored | Locking |
|---|---|---|---|---|---|---|---|
| ✓ | / | 24.61GB | 11.72GB | 12.88GB | 47% | true | |
| ✓ | /boot | 106.31MB | 19.87MB | 86.45MB | 18% | true | |
| ✓ | /home | 423.89GB | 379.00GB | 44.89GB | 89% | true | |
| ⚠ | Physical memory | 3.76GB | 3.72GB | 41.82MB | 98% | true | |
| ✓ | Swap space | 10.00GB | 1.43GB | 8.57GB | 14% | true | |

| Status | Severity | Event Class | Summary | First Seen | Last Seen | Count |
|---|---|---|---|---|---|---|
| | ⚠ | /Perf/Filesystem | disk space threshold: 98.8% used | 2010-12-21 12:58:44 | 2010-12-21 16:12:17 | 39 |

*Figure 5: Device details for lotschy.skills-1st.co.uk with events for exceeded thresholds*

### 4.1.2 The Zope Object Database (ZODB)

Zenoss is developed in Python using the open source Zope web application server – see http://www.zope.org/WhatIsZope for more information.

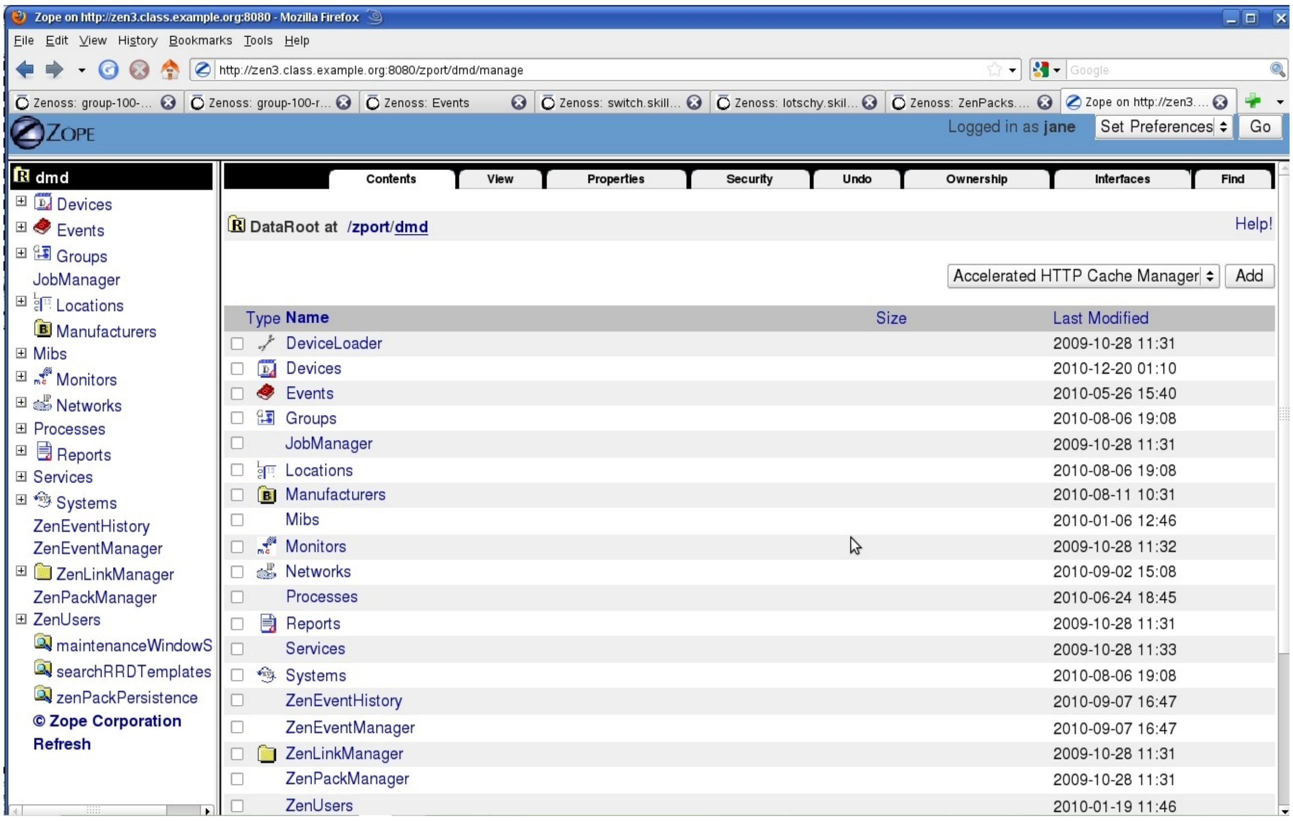
The Zope Object Database (ZODB) is an object-oriented Configuration Management Database (CMDB) used by Zope to store Python objects and their states; modeler plugins maintain information about devices and their configuration in the ZODB.

Zenoss uses ZEO, which is a layer between Zope and the ZODB. ZEO allows for multiple Zope servers to connect to the same ZODB. The ZODB is started and stopped by **zeoctl** . Note that the Zenoss documentation tends to use ZODB and ZEO rather interchangeably.

One way to get a feel for what is in the ZODB database and what Zope provides, is to use the Zope Management Interface (ZMI); point your browser at:

```
    http://<zenoss server>:8080/zport/dmd/manage
```

You will need to authenticate yourself as a Zenoss user with Manager privileges if you have not already done so. The resulting screens allow you to explore the Zenoss **objects** (such as devices, event classes and MIBs) and also to display the **instances** of those objects (such as switch.skills-1st.class.example.org and BRIDGE-MIB).


Zope on http://zen3.class.example.org:8080 - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://zen3.class.example.org:8080/zport/dmd/manage

Logged in as jane | Set Preferences | Go

dmd
- Devices
- Events
- Groups
- JobManager
- Locations
- Manufacturers
- Mibs
- Monitors
- Networks
- Processes
- Reports
- Services
- Systems
- ZenEventHistory
- ZenEventManager
- ZenLinkManager
- ZenPackManager
- ZenUsers
- maintenanceWindowS
- searchRRDTemplates
- zenPackPersistence

© Zope Corporation
Refresh

Contents | View | Properties | Security | Undo | Ownership | Interfaces | Find

DataRoot at /zport/dmd

Help!

Accelerated HTTP Cache Manager | Add

| Type | Name | Size | Last Modified |
|---|---|---|---|
| | DeviceLoader | | 2009-10-28 11:31 |
| | Devices | | 2010-12-20 01:10 |
| | Events | | 2010-05-26 15:40 |
| | Groups | | 2010-08-06 19:08 |
| | JobManager | | 2009-10-28 11:31 |
| | Locations | | 2010-08-06 19:08 |
| | Manufacturers | | 2010-08-11 10:31 |
| | Mibs | | 2010-01-06 12:46 |
| | Monitors | | 2009-10-28 11:32 |
| | Networks | | 2010-09-02 15:08 |
| | Processes | | 2010-06-24 18:45 |
| | Reports | | 2009-10-28 11:31 |
| | Services | | 2009-10-28 11:33 |
| | Systems | | 2010-08-06 19:08 |
| | ZenEventHistory | | 2010-09-07 16:47 |
| | ZenEventManager | | 2010-09-07 16:47 |
| | ZenLinkManager | | 2009-10-28 11:31 |
| | ZenPackManager | | 2009-10-28 11:31 |
| | ZenUsers | | 2010-01-19 11:46 |

*Figure 6: Accessing the ZODB database using the Zope interface*

The top level of the ZODB database is *zport/dmd* (where dmd stands for **Device Management Database**). Note that the Zenoss 3 Developer's Guide has a very helpful glossary at the back which explains many of Zope's terms. If you omit the *manage* from the URL shown above, you will simply get to the standard Zenoss dashboard; adding *manage* provides access to the underlying Zope. For more information on Zope and the Zope Management Interface (ZMI), see http://docs.zope.org/zope2/zope2book/index.html , especially Chapter 6, "Using the Zope Management Interface", http://docs.zope.org/zope2/zope2book/UsingZope.html .

### 4.1.3 Coding techniques and terminology

When developing ZenPack code (in fact when administering Zenoss in any way), always ensure you are logged on as the *zenoss* user. When Zenoss is installed, this user is created but will be setup such that you cannot login directly as zenoss; you need to su to root and then use:

```
        su - zenoss
```

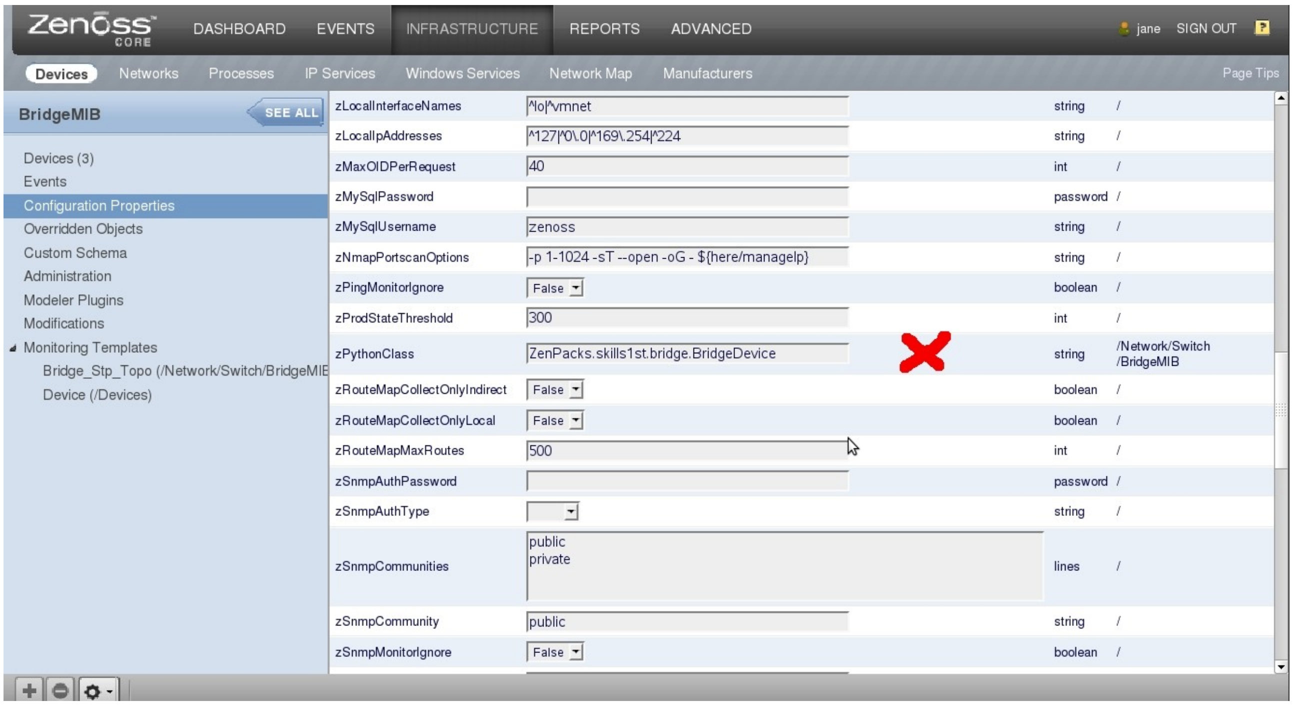
to switch to the zenoss user.

If Python code is to be written, be aware that Python is very white-space sensitive. Program constructs such as if-then-else, while loops, for loops and many other coding elements depend on white space indentation (and the same number of spaces for the same level of the construct). If testing Python with the Zenoss-provided **zendmd** utility, the same white-space rules must be obeyed.

If a ZenPack is going to support new types of devices then a new **Python object class** needs to be created to describe the unique features of this device type. As with all object-oriented code, the new class can (and probably should) inherit some characteristics from its parent object class in a class hierarchy. Thus, the ZenPack discussed in this paper will create a new device class called *BridgeMIB*, which inherits from the standard device class */Devices/Network/Switch*; the unique characteristics of such a device are coded in a Python file in the base directory of the ZenPack (*/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/BridgeDevice.py*).

A new device class is associated with a Python class through the *zPythonClass* zProperty (note that you do **not** specify the zPythonClass as a normal filesystem path but as a dotted class path from the ZenPack ie. *ZenPacks.skills1st.bridge.BridgeDevice* represents the file *BridgeDevice.py* (but don't include the *.py* ) under the Zenoss ZenPacks directory *ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge* . More details on this later.

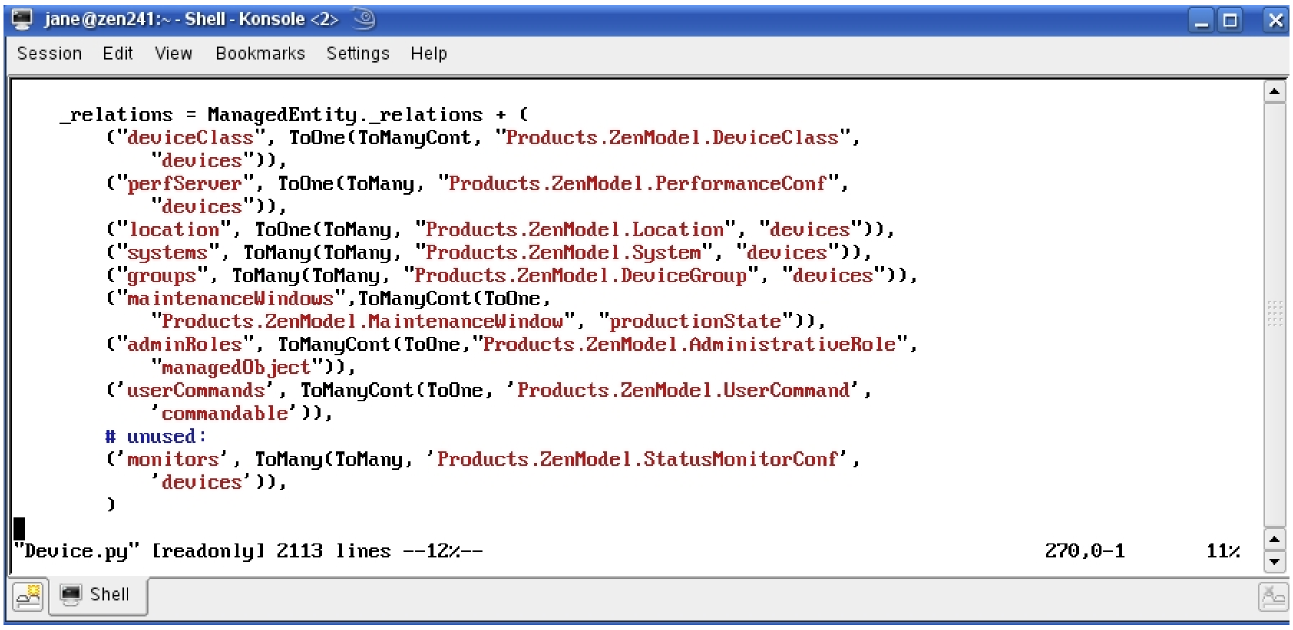
Standard object classes, such as *Device*, *OSComponent* and *IpInterface* can be found under *$ZENHOME/Products/ZenModel* . Note that each object class definition will have two files. The *.py* file is the Python source code; the *.pyc* file is the compiled Python code. There is no need to manually compile any Python code for Zenoss as this will be done automatically, as required.



| | | | |
|---|---|---|---|
| zLocalInterfaceNames | ^lo\|^vmnet | string | / |
| zLocalIpAddresses | ^127\|^0\.0\|^169\.254\|^224 | string | / |
| zMaxOIDPerRequest | 40 | int | / |
| zMySqlPassword | | password | / |
| zMySqlUsername | zenoss | string | / |
| zNmapPortscanOptions | -p 1-1024 -sT --open -oG - ${here/manageIp} | string | / |
| zPingMonitorIgnore | False | boolean | / |
| zProdStateThreshold | 300 | int | / |
| zPythonClass | ZenPacks.skills1st.bridge.BridgeDevice | string | /Network/Switch /BridgeMIB |
| zRouteMapCollectOnlyIndirect | False | boolean | / |
| zRouteMapCollectOnlyLocal | False | boolean | / |
| zRouteMapMaxRoutes | 500 | int | / |
| zSnmpAuthPassword | | password | / |
| zSnmpAuthType | | string | / |
| zSnmpCommunities | public<br>private | lines | / |
| zSnmpCommunity | public | string | / |
| zSnmpMonitorIgnore | False | boolean | / |

*Figure 7: Associating a device class (BridgeMIB) with a Python class*

Object classes that represent devices can have **relationships** with other classes. For example, *$ZENHOME/Products/ZenModel/Device.py,* which defines the base object class for devices, specifies a number of relationships as shown in Figure 8.


jane@zen241:~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
Shell

```
    _relations = ManagedEntity._relations + (
        ("deviceClass", ToOne(ToManyCont, "Products.ZenModel.DeviceClass",
            "devices")),
        ("perfServer", ToOne(ToMany, "Products.ZenModel.PerformanceConf",
            "devices")),
        ("location", ToOne(ToMany, "Products.ZenModel.Location", "devices")),
        ("systems", ToMany(ToMany, "Products.ZenModel.System", "devices")),
        ("groups", ToMany(ToMany, "Products.ZenModel.DeviceGroup", "devices")),
        ("maintenanceWindows",ToManyCont(ToOne,
            "Products.ZenModel.MaintenanceWindow", "productionState")),
        ("adminRoles", ToManyCont(ToOne,"Products.ZenModel.AdministrativeRole",
            "managedObject")),
        ('userCommands', ToManyCont(ToOne, 'Products.ZenModel.UserCommand',
            'commandable')),
        # unused:
        ('monitors', ToMany(ToMany, 'Products.ZenModel.StatusMonitorConf',
            'devices')),
        )

"Device.py" [readonly] 2113 lines --12%--                    270,0-1       11%
```

*Figure 8: Relationships defined in Device.py*

Look at Chapter 9 of the Zenoss 3 Developer's Guide for details on the different types of relationships. Fundamentally, the code shown in Figure 8 is saying:

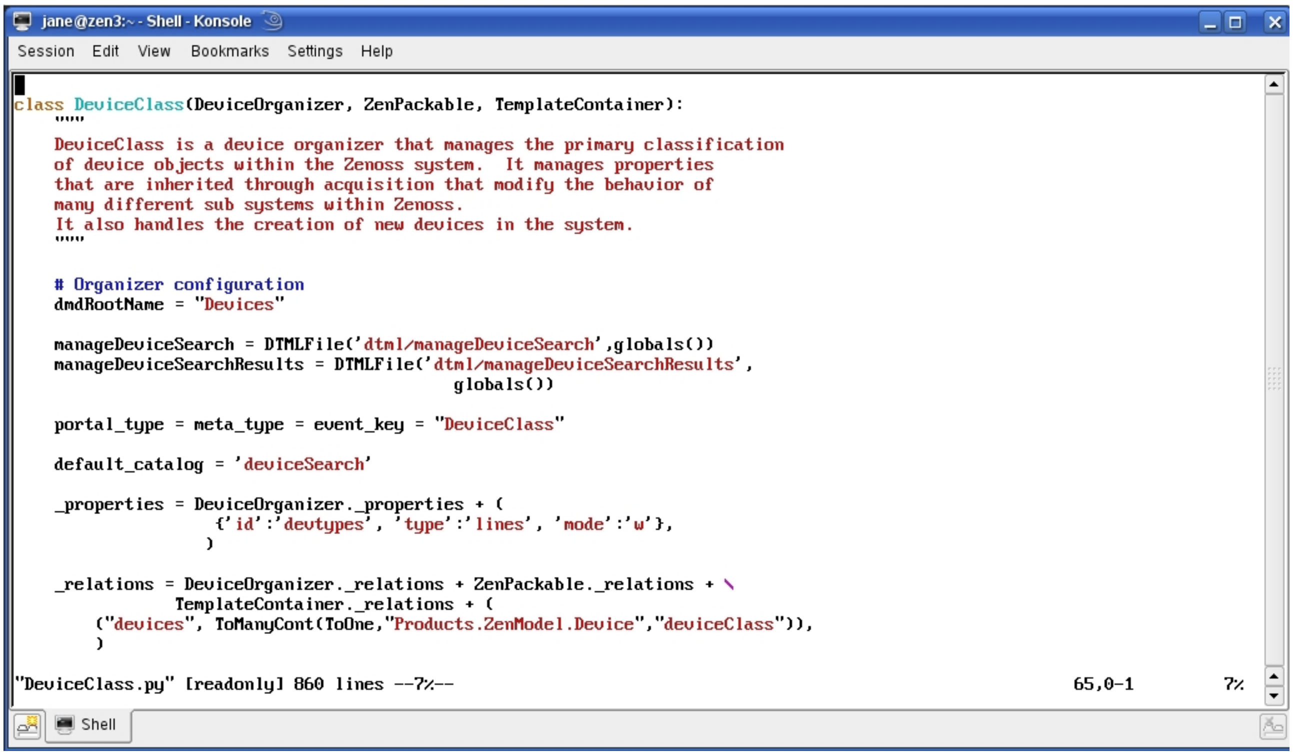
- A Device has a ToOne relationship with the object class DeviceClass (ie. any specific device can only belong to one device class)
- A Device has a ToOne relationship with the object class PerformanceConf (ie. any specific device will have only one performance data collector associated with it)
- A Device has a ToOne relationship with the object class Location (ie. any specific device can only be assigned to a single location)
- A Device has a ToMany relationship with the object class System (ie. any specific device can be assigned to several System groupings)
- A Device has a ToMany relationship with the object class DeviceGroup (ie. any specific device can be assigned to several Groups)
- A Device has a ToManyCont relationship with the object class MaintenanceWindow (ie. any specific device can contain several maintenance windows)

- A Device has a ToManyCont relationship with the object class AdministrativeRole (ie. any specific device can contain several administrative roles)
- A Device has a ToManyCont relationship with the object class UserCommand (ie. any specific device can contain several user commands)
- A Device has a ToMany relationship with the object class StatusMonitorConf (ie. any specific device can have several status monitors associated with it)

The syntax of the relationship statement seems rather perverse. Taking the first relationship from Device.py as an example:

```
("deviceClass", ToOne(ToManyCont, "Products.ZenModel.DeviceClass", "devices"))
```

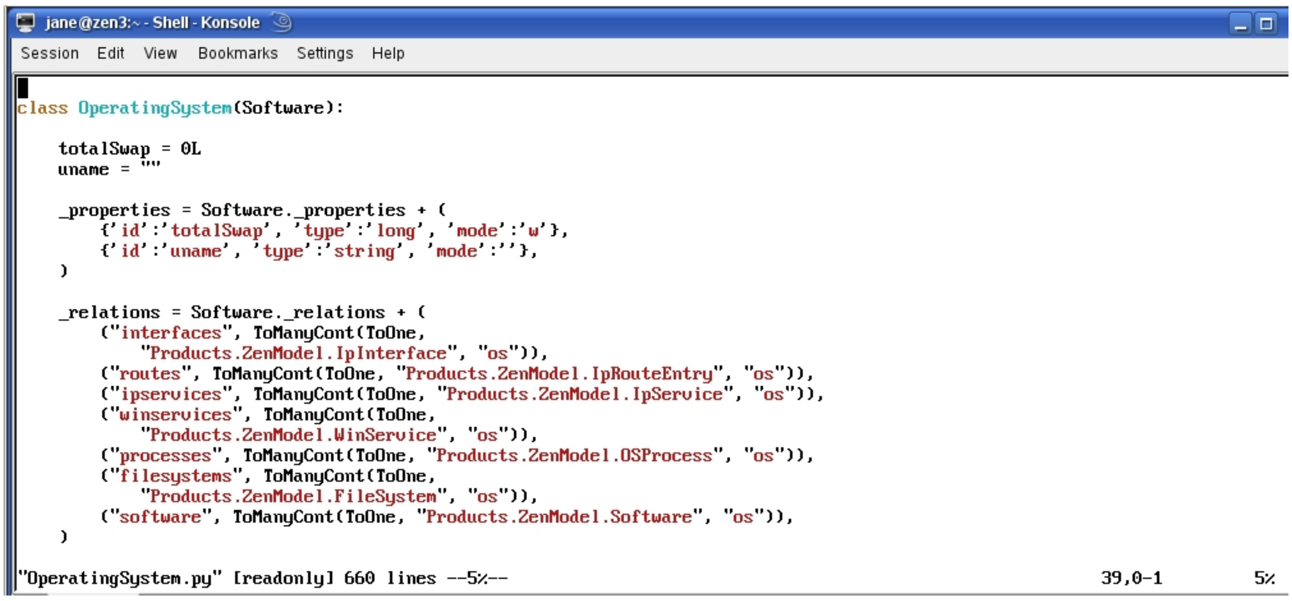
- All relationships in this file are for the object class being defined, ie. Device in $ZENHOME/Products/ZenModel/Device.py
- The first field, *deviceClass* is the name of **this** relationship
- The relationship is a ToOne between Device and DeviceClass
- There is a corresponding relationship between DeviceClass and Device
    - The file $ZENHOME/Products/ZenModel/DeviceClass.py must contain this corresponding relationship (see Figure 9 )
    - The relationship is a ToManyCont ie. a DeviceClass can contain many devices
    - The name of the relationship defined in $ZENHOME/Products/ZenModel/DeviceClass.py is the last field, ie. *devices*


jane@zen3:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
class DeviceClass(DeviceOrganizer, ZenPackable, TemplateContainer):
    """
    DeviceClass is a device organizer that manages the primary classification
    of device objects within the Zenoss system.  It manages properties
    that are inherited through acquisition that modify the behavior of
    many different sub systems within Zenoss.
    It also handles the creation of new devices in the system.
    """

    # Organizer configuration
    dmdRootName = "Devices"

    manageDeviceSearch = DTMLFile('dtml/manageDeviceSearch',globals())
    manageDeviceSearchResults = DTMLFile('dtml/manageDeviceSearchResults',
                                        globals())

    portal_type = meta_type = event_key = "DeviceClass"

    default_catalog = 'deviceSearch'

    _properties = DeviceOrganizer._properties + (
                {'id':'devtypes', 'type':'lines', 'mode':'w'},
               )

    _relations = DeviceOrganizer._relations + ZenPackable._relations + \
                TemplateContainer._relations + (
        ("devices", ToManyCont(ToOne,"Products.ZenModel.Device","deviceClass")),
        )

"DeviceClass.py" [readonly] 860 lines --7%--                    65,0-1        7%
```

Figure 9: $ZENHOME/Products/ZenModel/DeviceClass.py showing corresponding relationship with Device

Note that there is a specific relationship type when an object **contains** another object. Better examples exist in $ZENHOME/Products/ZenModel/OperatingSystem.py where an Operating System may contain many interfaces, routes, ipservices, winservices, processes, filesystems and software packages.

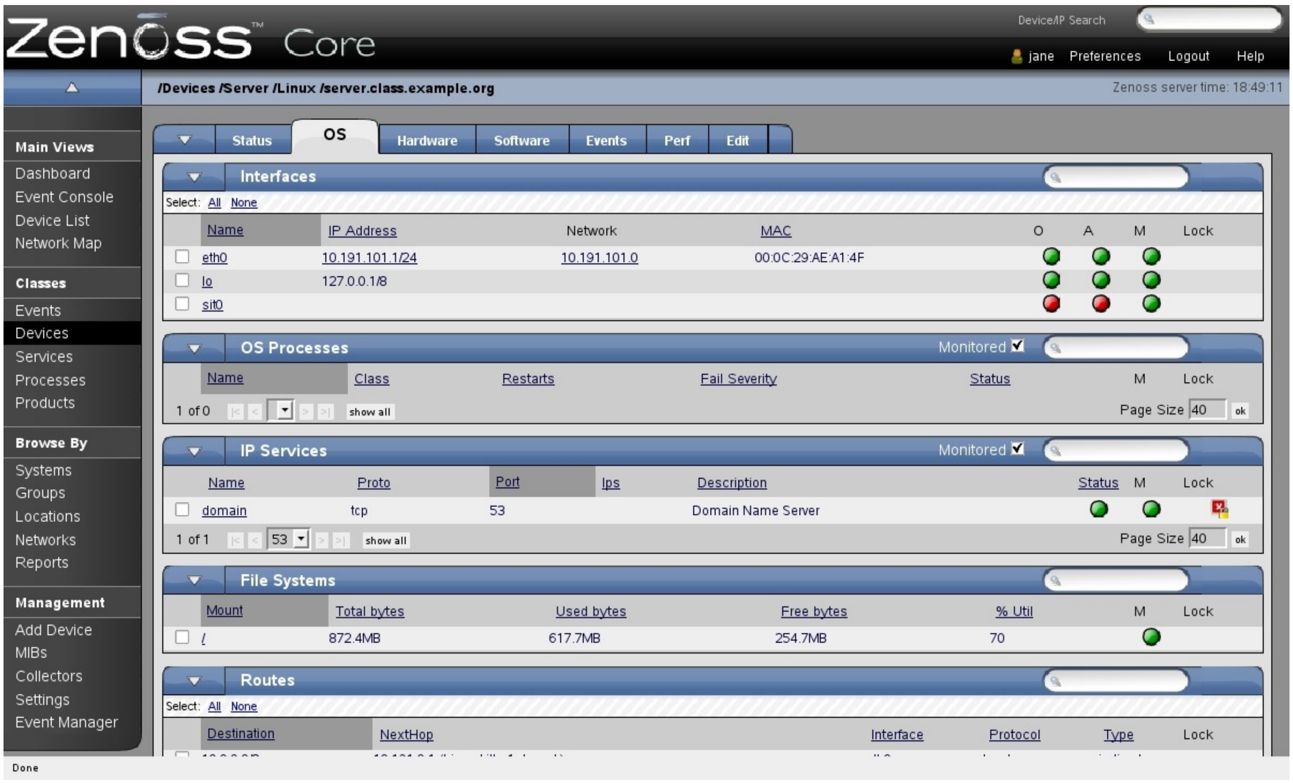

jane@zen3:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
class OperatingSystem(Software):

    totalSwap = 0L
    uname = ""

    _properties = Software._properties + (
        {'id':'totalSwap', 'type':'long', 'mode':'w'},
        {'id':'uname', 'type':'string', 'mode':''},
    )

    _relations = Software._relations + (
        ("interfaces", ToManyCont(ToOne,
            "Products.ZenModel.IpInterface", "os")),
        ("routes", ToManyCont(ToOne, "Products.ZenModel.IpRouteEntry", "os")),
        ("ipservices", ToManyCont(ToOne, "Products.ZenModel.IpService", "os")),
        ("winservices", ToManyCont(ToOne,
            "Products.ZenModel.WinService", "os")),
        ("processes", ToManyCont(ToOne, "Products.ZenModel.OSProcess", "os")),
        ("filesystems", ToManyCont(ToOne,
            "Products.ZenModel.FileSystem", "os")),
        ("software", ToManyCont(ToOne, "Products.ZenModel.Software", "os")),
    )

"OperatingSystem.py" [readonly] 660 lines --5%--                  39,0-1        5%
```

Figure 10: ToManyCont relationships for the OperatingSystem object class

Where a container relationship exists, this leads to a requirement to be able to conveniently display data about those contained components. Almost all devices will have some contained components – a network interface, perhaps. Other devices such as servers may have lots of components.

The Zenoss 2 GUI displayed device components under an *OS* tab as shown in Figure 11. Clicking on a component, such as the *eth0* interface, produced panels with component configuration information and performance graphs.



/Devices /Server /Linux /server.class.example.org

**Interfaces**

Select: All None

| | Name | IP Address | Network | MAC | O | A | M | Lock |
|---|---|---|---|---|---|---|---|---|
| ☐ | eth0 | 10.191.101.1/24 | 10.191.101.0 | 00:0C:29:AE:A1:4F | 🟢 | 🟢 | 🟢 | |
| ☐ | lo | 127.0.0.1/8 | | | 🟢 | 🟢 | 🟢 | |
| ☐ | sit0 | | | | 🔴 | 🔴 | 🟢 | |

**OS Processes** Monitored ☑

| Name | Class | Restarts | Fail Severity | Status | M | Lock |
|---|---|---|---|---|---|---|

1 of 0 show all Page Size 40 ok

**IP Services** Monitored ☑

| | Name | Proto | Port | Ips | Description | Status | M | Lock |
|---|---|---|---|---|---|---|---|---|
| ☐ | domain | tcp | 53 | | Domain Name Server | 🟢 | 🟢 | |

1 of 1 53 show all Page Size 40 ok

**File Systems**

| | Mount | Total bytes | Used bytes | Free bytes | % Util | M | Lock |
|---|---|---|---|---|---|---|---|
| ☐ | / | 872.4MB | 617.7MB | 254.7MB | 70 | 🟢 | |

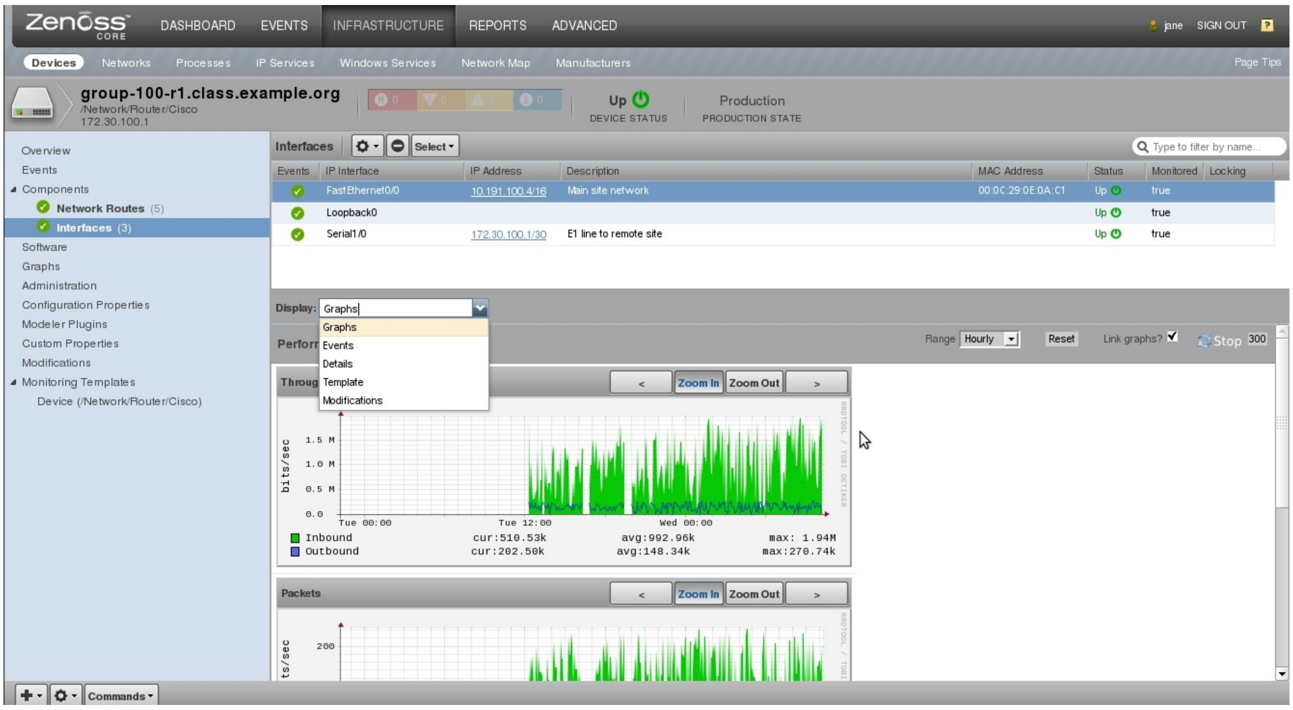
**Routes**

Select: All None

| Destination | NextHop | Interface | Protocol | Type | Lock |
|---|---|---|---|---|---|

*Figure 11: Zenoss 2 GUI with OS tab for device components*

In the Zenoss 3 GUI, device components are shown in the left-hand menu of a device's details page, with a sub-menu for each component type, as shown in Figure 12. Clicking a component type results in a panel showing all instances of that component. In the middle of this panel is a *Display* dropdown which is customisable but typically offers *Graphs*, *Events, Template* and *Details* menus for the selected component instance.


group-100-r1.class.example.org
/Network/Router/Cisco
172.30.100.1

Up — DEVICE STATUS
Production — PRODUCTION STATE

| Events | IP Interface | IP Address | Description | MAC Address | Status | Monitored | Locking |
|---|---|---|---|---|---|---|---|
| ✓ | FastEthernet0/0 | 10.191.100.4/16 | Main site network | 00:0C:29:0E:0A:C1 | Up | true | |
| ✓ | Loopback0 | | | | Up | true | |
| ✓ | Serial1/0 | 172.30.100.1/30 | E1 line to remote site | | Up | true | |

Display: Graphs

Range: Hourly    Link graphs? ✓    300

```
Inbound     cur:510.53k    avg:992.96k    max: 1.94M
Outbound    cur:202.50k    avg:148.34k    max:270.74k
```

*Figure 12: The Components option for a device showing data for the contained Interfaces relationship*

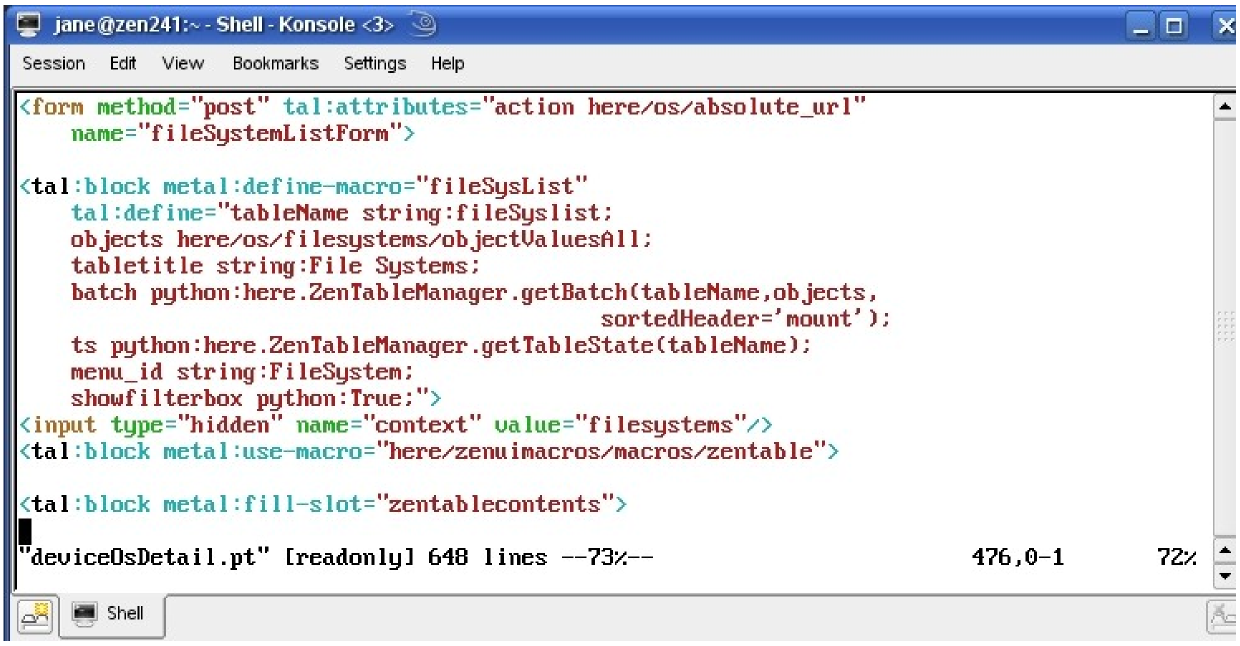
Prior to Zenoss 3, most web pages for displaying data were defined in **skins** files. The skins files for the standard Zenoss objects are in the *$ZENHOME/Products/ZenModel/skins/zenmodel* directory and all have a *.pt* file extension (for Page Template).

With Zenoss 3 there is now a move towards using **JavaScript** to define page layouts so we are currently in a transition period where both skins files and JavaScript files are employed. Chapter 14 of the Zenoss 3 Developer's Guide has some information on conversion tasks when moving from Zenoss 2 to Zenoss 3.

The Zenoss 3 Developer's Guide, Chapter 13 provides details on writing skins files. You can use a mixture of:

- HyperText Markup Language (HTML)
- Cascading Style Sheets (CSS)
- Zope 2, Zope Page Templates (ZPT) and the Template Attribute Language (TAL)
- ZPT and Macro Expansion for TAL (METAL)
- JavaScript / **A**synchronous **J**avaScript **A**nd **X**ML (AJAX))
- Yahoo User Interface (YUI) Library and Mochikit

The file that defines the page for the Zenoss 2 OS tab, shown above in Figure 11, is *$ZENHOME/Products/ZenModel/skins/zenmodel/deviceOsDetail.pt*. It defines a form containing a table for each type of component, where the data to populate the table comes from the ZODB database.


jane@zen241:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
Shell

```
<form method="post" tal:attributes="action here/os/absolute_url"
    name="fileSystemListForm">

<tal:block metal:define-macro="fileSysList"
    tal:define="tableName string:fileSyslist;
    objects here/os/filesystems/objectValuesAll;
    tabletitle string:File Systems;
    batch python:here.ZenTableManager.getBatch(tableName,objects,
                                          sortedHeader='mount');
    ts python:here.ZenTableManager.getTableState(tableName);
    menu_id string:FileSystem;
    showfilterbox python:True;">
<input type="hidden" name="context" value="filesystems"/>
<tal:block metal:use-macro="here/zenuimacros/macros/zentable">

<tal:block metal:fill-slot="zentablecontents">

"deviceOsDetail.pt" [readonly] 648 lines --73%--                    476,0-1        72%
```

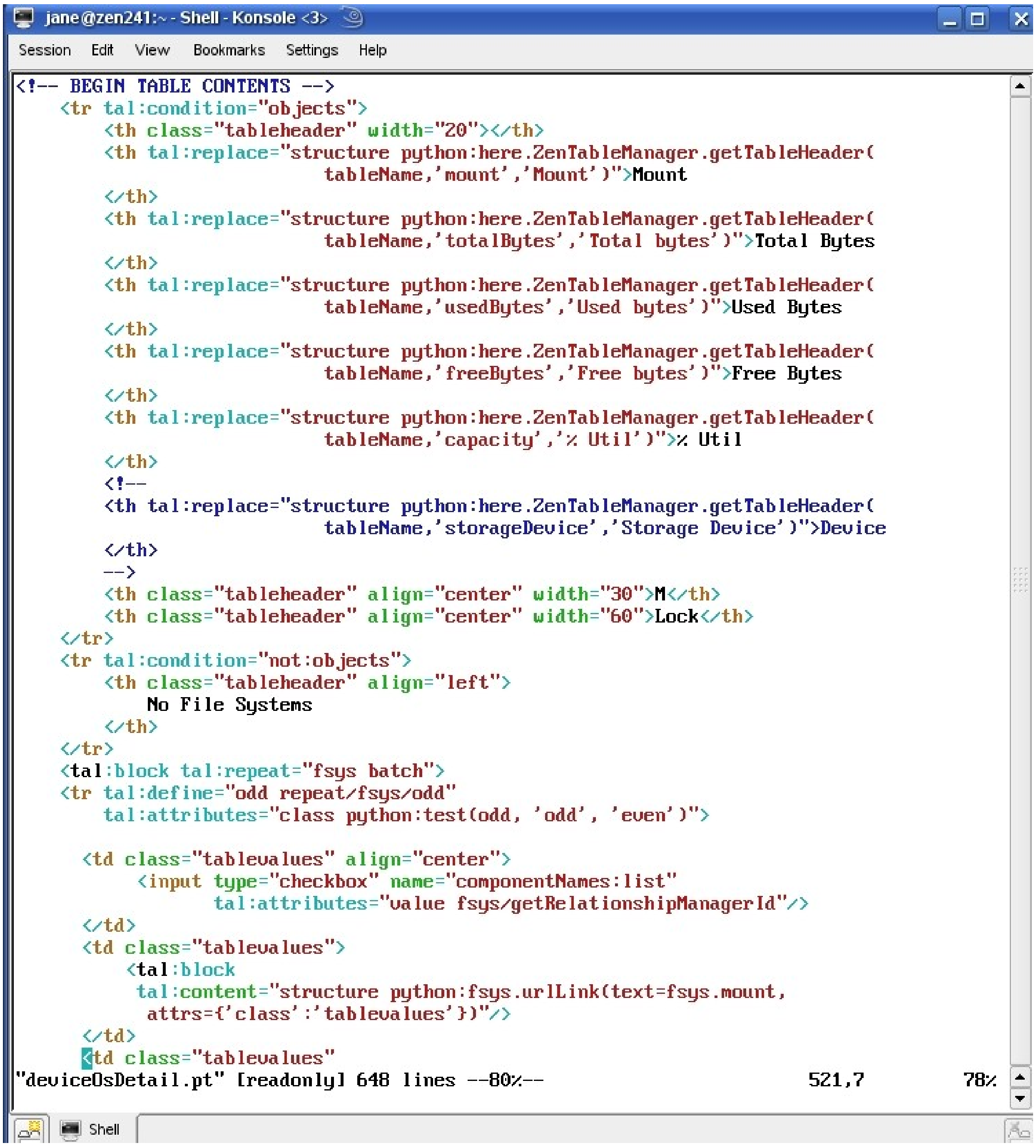
*Figure 13: deviceOsDetail.pt skin file with definition of form for displaying filesystem information*

The key line to note in Figure 13 is:

```
    objects here/os/filesystem/objectValuesAll;
```

where *here* is the device in question (such as server.class.example.org), *os* is the Operating System object on the device, which in turn contains the *filesystem* object. *objectValuesAll* will return a table of data with one row for each filesystem on the device.

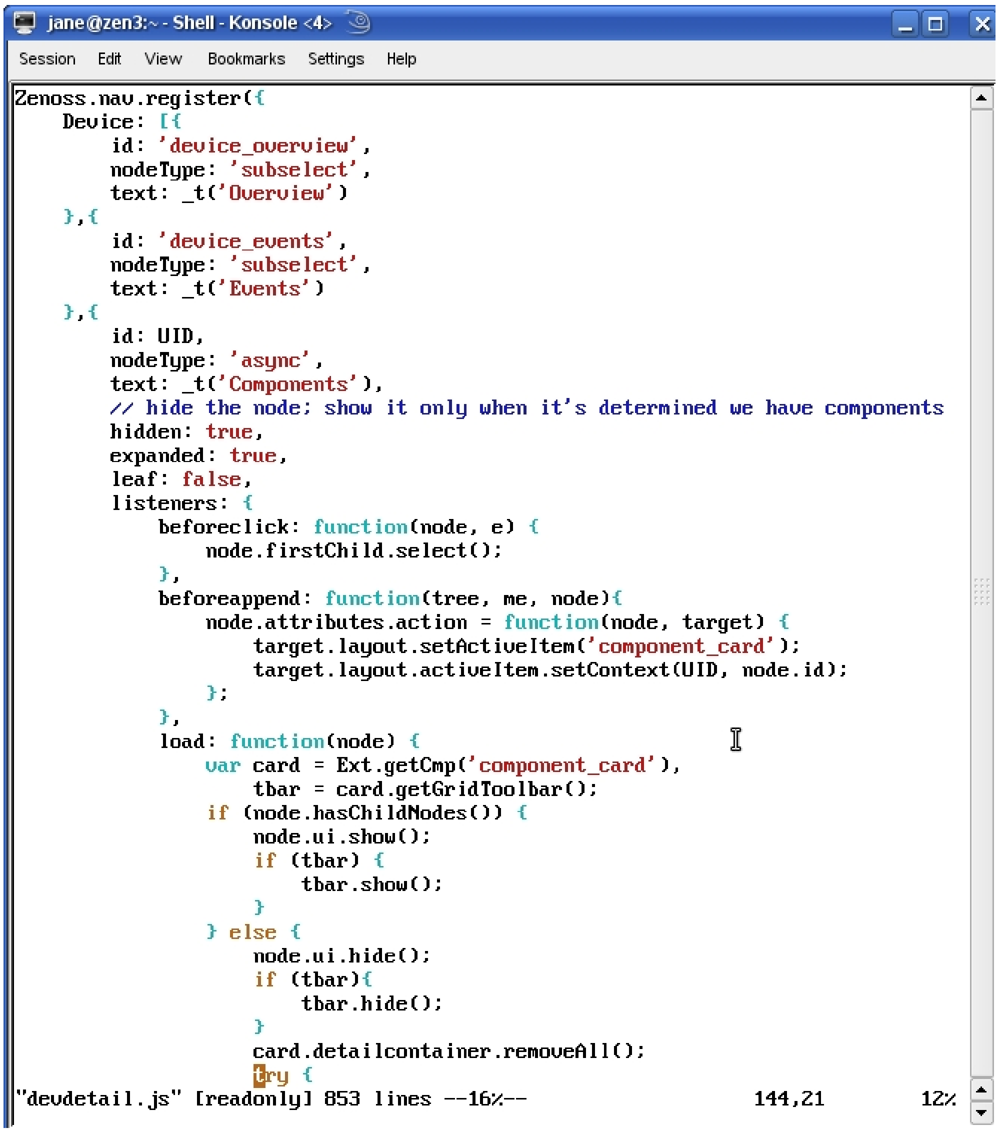
The layout of the table, including header columns and data columns can be very finely controlled. The first half of Figure 14 defines the table header columns; the middle 5 lines shows a check to ensure that data does actually exist to display; the next 3 lines (with *odd* and *even* in them) ensures that the rows of the table will have alternating light and dark backgrounds; and the rest of the screenshot is the start of the data values to populate the filesystem table. The intricacies of skins files will be examined in more detail later.


jane@zen241:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
Shell

```
<!-- BEGIN TABLE CONTENTS -->
    <tr tal:condition="objects">
        <th class="tableheader" width="20"></th>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(
                            tableName,'mount','Mount')">Mount
        </th>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(
                            tableName,'totalBytes','Total bytes')">Total Bytes
        </th>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(
                            tableName,'usedBytes','Used bytes')">Used Bytes
        </th>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(
                            tableName,'freeBytes','Free bytes')">Free Bytes
        </th>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(
                            tableName,'capacity','% Util')">% Util
        </th>
        <!--
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(
                            tableName,'storageDevice','Storage Device')">Device
        </th>
        -->
        <th class="tableheader" align="center" width="30">M</th>
        <th class="tableheader" align="center" width="60">Lock</th>
    </tr>
    <tr tal:condition="not:objects">
        <th class="tableheader" align="left">
            No File Systems
        </th>
    </tr>
    <tal:block tal:repeat="fsys batch">
    <tr tal:define="odd repeat/fsys/odd"
        tal:attributes="class python:test(odd, 'odd', 'even')">

      <td class="tablevalues" align="center">
           <input type="checkbox" name="componentNames:list"
                  tal:attributes="value fsys/getRelationshipManagerId"/>
      </td>
      <td class="tablevalues">
          <tal:block
           tal:content="structure python:fsys.urlLink(text=fsys.mount,
            attrs={'class':'tablevalues'})"/>
      </td>
      <td class="tablevalues"
"deviceOsDetail.pt" [readonly] 648 lines --80%--                 521,7          78%
```

*Figure 14: deviceOsDetail.pt showing layout of table for filesystems data*

In Zenoss 3 the devices main page has been completely redesigned and uses JavaScript to present device details, components and subsequent component details and graphs. A new subdirectory, **ZenUI3**, appears under *$ZENHOME/Products* which contains a large subdirectory hierarchy with JavaScript files. The Device Detail page is defined in *$ZENHOME/Products/ZenUI3/browser/resources/js/zenoss/devdetail.js* – this presents the overall view for a device.

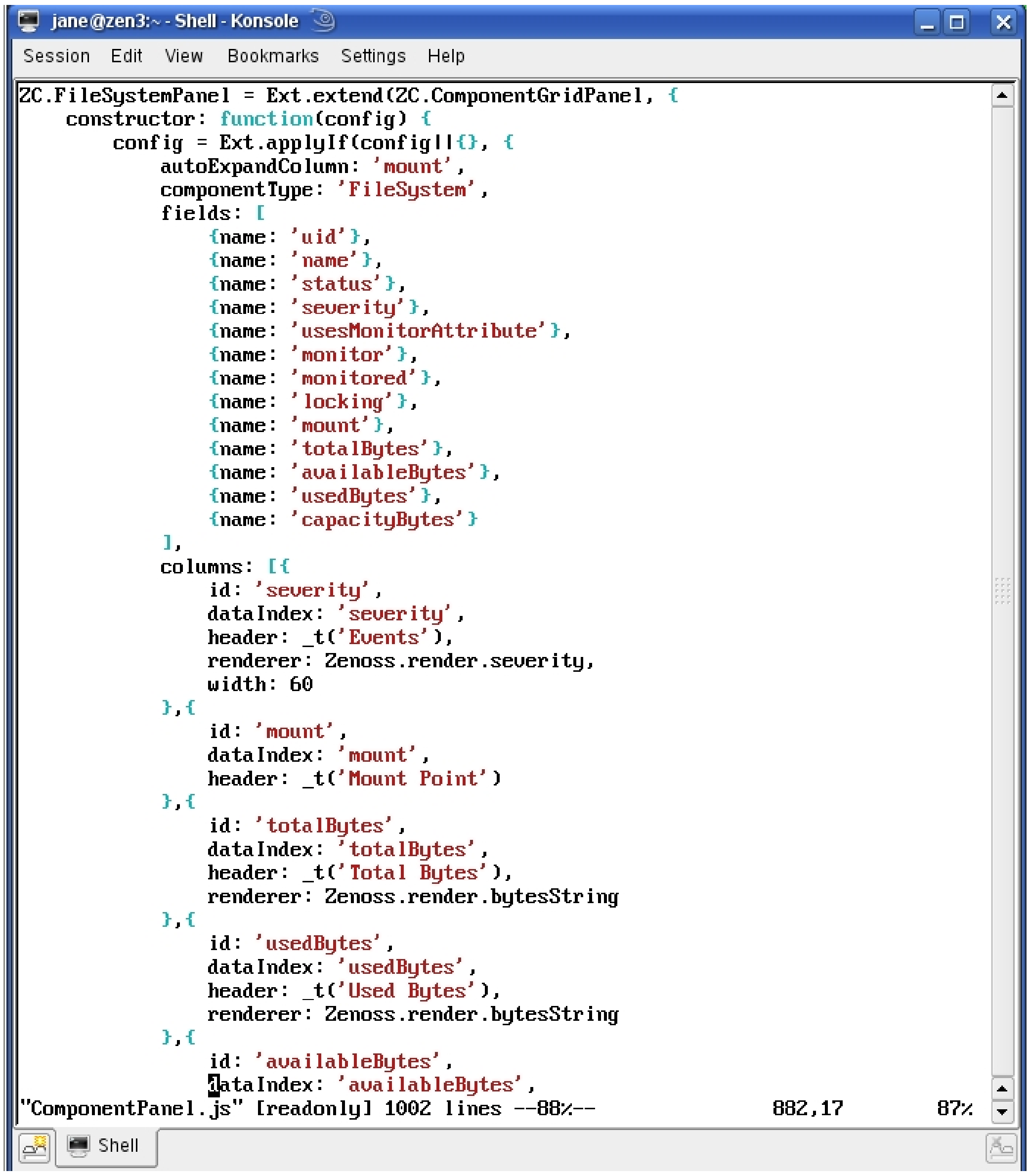

jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
Zenoss.nav.register({
    Device: [{
        id: 'device_overview',
        nodeType: 'subselect',
        text: _t('Overview')
    },{
        id: 'device_events',
        nodeType: 'subselect',
        text: _t('Events')
    },{
        id: UID,
        nodeType: 'async',
        text: _t('Components'),
        // hide the node; show it only when it's determined we have components
        hidden: true,
        expanded: true,
        leaf: false,
        listeners: {
            beforeclick: function(node, e) {
                node.firstChild.select();
            },
            beforeappend: function(tree, me, node){
                node.attributes.action = function(node, target) {
                    target.layout.setActiveItem('component_card');
                    target.layout.activeItem.setContext(UID, node.id);
                };
            },
            load: function(node) {
                var card = Ext.getCmp('component_card'),
                    tbar = card.getGridToolbar();
                if (node.hasChildNodes()) {
                    node.ui.show();
                    if (tbar) {
                        tbar.show();
                    }
                } else {
                    node.ui.hide();
                    if (tbar){
                        tbar.hide();
                    }
                    card.detailcontainer.removeAll();
                    try {
"devdetail.js" [readonly] 853 lines --16%--                    144,21        12%
```

*Figure 15: devdetail.js fragment showing some of the navigation menu options*

Component details display is handled by *$ZENHOME/Products/ZenUI3/browser/resources/js/zenoss/ComponentPanel.js* .

Comparison of the Page Template code in Figure 14 and the JavaScript in Figure 16 shows similar headers.

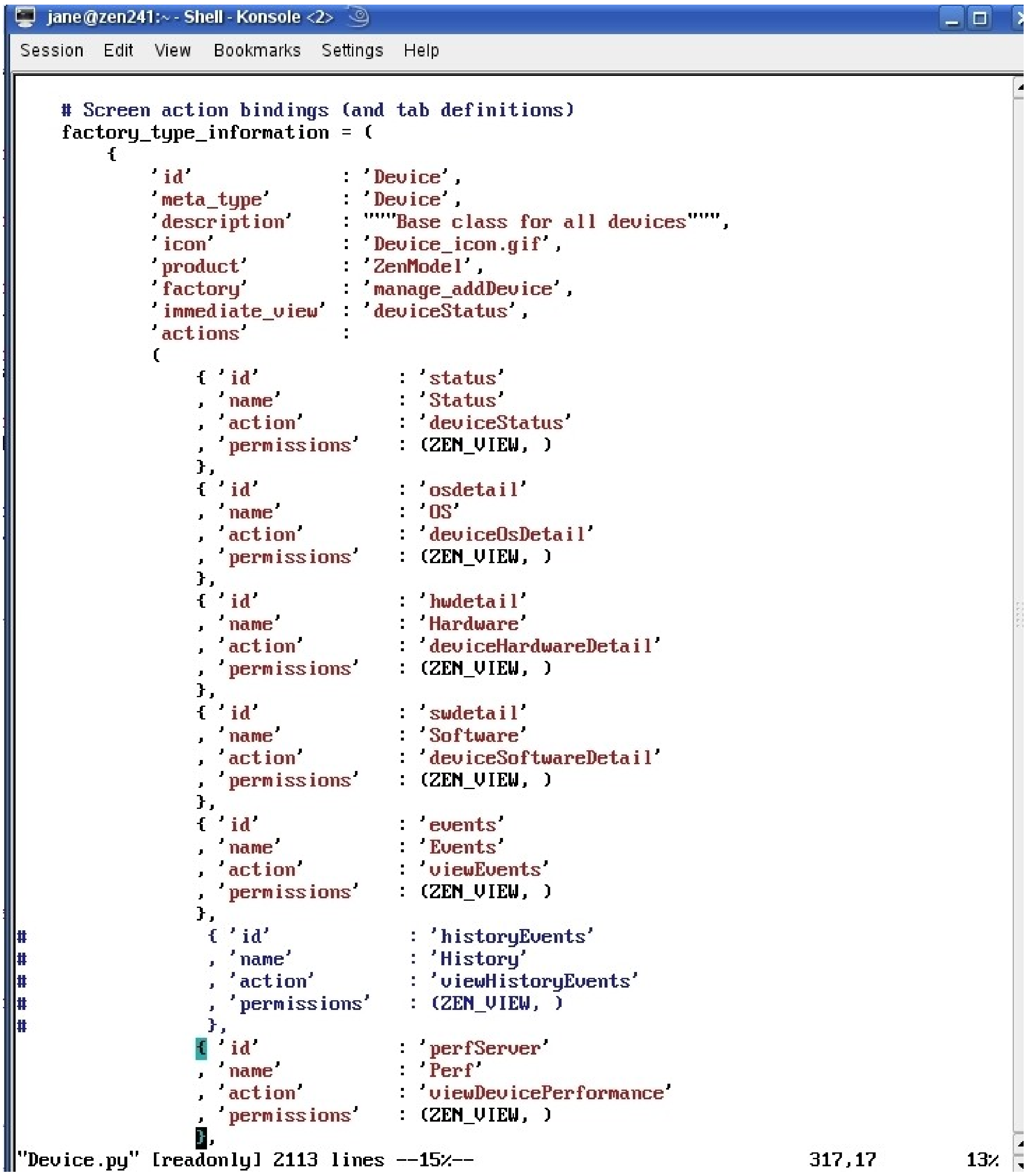

jane@zen3:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
ZC.FileSystemPanel = Ext.extend(ZC.ComponentGridPanel, {
    constructor: function(config) {
        config = Ext.applyIf(config||{}, {
            autoExpandColumn: 'mount',
            componentType: 'FileSystem',
            fields: [
                {name: 'uid'},
                {name: 'name'},
                {name: 'status'},
                {name: 'severity'},
                {name: 'usesMonitorAttribute'},
                {name: 'monitor'},
                {name: 'monitored'},
                {name: 'locking'},
                {name: 'mount'},
                {name: 'totalBytes'},
                {name: 'availableBytes'},
                {name: 'usedBytes'},
                {name: 'capacityBytes'}
            ],
            columns: [{
                id: 'severity',
                dataIndex: 'severity',
                header: _t('Events'),
                renderer: Zenoss.render.severity,
                width: 60
            },{
                id: 'mount',
                dataIndex: 'mount',
                header: _t('Mount Point')
            },{
                id: 'totalBytes',
                dataIndex: 'totalBytes',
                header: _t('Total Bytes'),
                renderer: Zenoss.render.bytesString
            },{
                id: 'usedBytes',
                dataIndex: 'usedBytes',
                header: _t('Used Bytes'),
                renderer: Zenoss.render.bytesString
            },{
                id: 'availableBytes',
                dataIndex: 'availableBytes',
"ComponentPanel.js" [readonly] 1002 lines --88%--           882,17         87%
```

*Figure 16: Definition of FileSystem component display in ComponentPanel.js*

Again, JavaScript files will be revisited in much more detail later in this paper.

So what links the device object class with the skins file or JavaScript files that display web pages of data relating to a device? This is coded in **factory** statements in the object class file, after the relationship statements. Each tab required for the object in Zenoss 2 has a stanza defining its *id*, *name*, *action* and *permissions;* it is the *action* field that specifies the name of the skins file (without the *.pt* ). Compare the (incomplete) definitions in Figure 17 with the tabs shown for a device in Figure 11. The *name* field gives the name shown on the tab; the *action* field should match with the name of a skins file (without the *.pt* ) in *$ZENHOME/Products/ZenModel/skins/zenmodel.*


jane@zen241:~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help

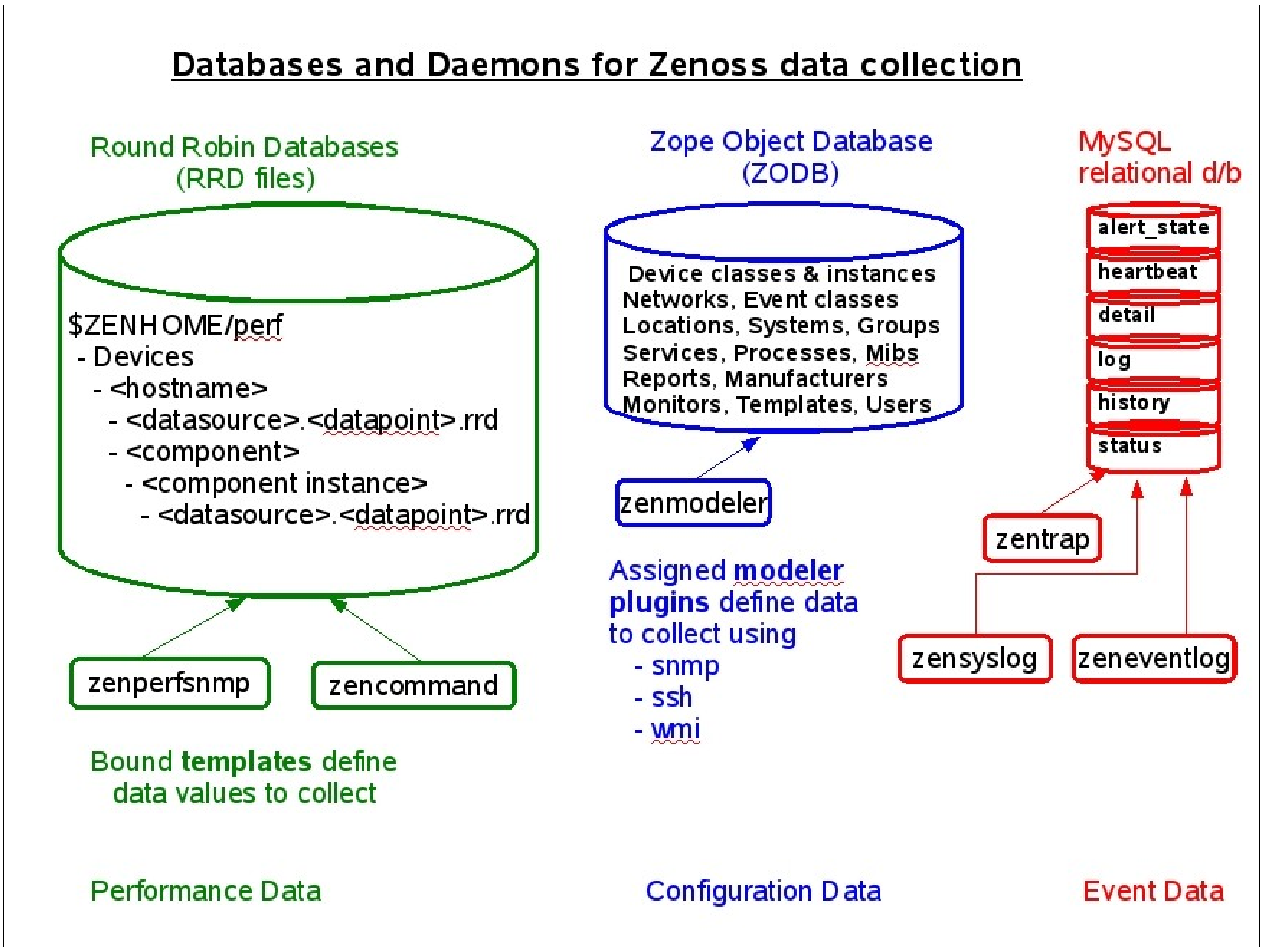
```
    # Screen action bindings (and tab definitions)
    factory_type_information = (
        {
            'id'             : 'Device',
            'meta_type'      : 'Device',
            'description'    : """Base class for all devices""",
            'icon'           : 'Device_icon.gif',
            'product'        : 'ZenModel',
            'factory'        : 'manage_addDevice',
            'immediate_view' : 'deviceStatus',
            'actions'        :
            (
                { 'id'            : 'status'
                , 'name'          : 'Status'
                , 'action'        : 'deviceStatus'
                , 'permissions'   : (ZEN_VIEW, )
                },
                { 'id'            : 'osdetail'
                , 'name'          : 'OS'
                , 'action'        : 'deviceOsDetail'
                , 'permissions'   : (ZEN_VIEW, )
                },
                { 'id'            : 'hwdetail'
                , 'name'          : 'Hardware'
                , 'action'        : 'deviceHardwareDetail'
                , 'permissions'   : (ZEN_VIEW, )
                },
                { 'id'            : 'swdetail'
                , 'name'          : 'Software'
                , 'action'        : 'deviceSoftwareDetail'
                , 'permissions'   : (ZEN_VIEW, )
                },
                { 'id'            : 'events'
                , 'name'          : 'Events'
                , 'action'        : 'viewEvents'
                , 'permissions'   : (ZEN_VIEW, )
                },
#                { 'id'            : 'historyEvents'
#                , 'name'          : 'History'
#                , 'action'        : 'viewHistoryEvents'
#                , 'permissions'   : (ZEN_VIEW, )
#                },
                { 'id'            : 'perfServer'
                , 'name'          : 'Perf'
                , 'action'        : 'viewDevicePerformance'
                , 'permissions'   : (ZEN_VIEW, )
                },
"Device.py" [readonly] 2113 lines --15%--                       317,17        13%
```

*Figure 17: Device.py object class file for Zenoss 2 showing the action filenames for each tab*

Device.py for Zenoss 3 has a very similar section with a few subtle changes; for example, the tab labelled *Perf* becomes a menu labelled *Graphs*, but largely the definitions are the same. The new mechanism in Zenoss 3 picks up the V2 Page Template (.pt) definitions, excludes a few specific tab names (such as *edit* and *events*) and then uses these old definitions to augment the new, standard Zenoss 3 left-hand menus for a device. Thus the Software and Graphs menus are added.

### 4.1.4 Databases, Daemons and Directories

To summarise this “Basic Principles” section, here are a couple of diagrams showing the architecture of Zenoss.



Databases and Daemons for Zenoss data collection
Round Robin Databases (RRD files)
$ZENHOME/perf
- Devices
- <hostname>
- <datasource>.<datapoint>.rrd
- <component>
- <component instance>
- <datasource>.<datapoint>.rrd
zenperfsnmp
zencommand
Bound templates define data values to collect
Performance Data
Zope Object Database (ZODB)
Device classes & instances
Networks, Event classes
Locations, Systems, Groups
Services, Processes, Mibs
Reports, Manufacturers
Monitors, Templates, Users
zenmodeler
Assigned modeler plugins define data to collect using
- snmp
- ssh
- wmi
Configuration Data
MySQL relational d/b
alert_state
heartbeat
detail
log
history
status
zentrap
zensyslog
zeneventlog
Event Data

*Figure 18: Databases and Daemons for Zenoss data collection*

Figure 18 shows the 3 different databases used by Zenoss:

- Performance data is held in Round Robin Database (RRD) files under $ZENHOME/perf
- Configuration data is held in the Zope Object Database (ZODB)
- Event data is held in a MySQL database

Performance data is typically collected at frequent intervals (SNMP data is collected every 5 minutes, by default). Templates define **datasources** and **datapoints** to be collected, where a datasource includes the source type (such as SNMP) and the OID to collect (in the case of SNMP). For SNMP data, the datapoint will have the same name as the datasource. If data is collected using ssh then the datasource type will be *COMMAND* and the polling interval can also be specified. Since an ssh command may return several datapoints, each has to be specified with a unique name. SNMP performance data is collected by the **zenperfsnmp** daemon whilst ssh data is collected by **zencommand**.

Performance data is stored under $ZENHOME/perf/Devices with a separate directory for each device. Performance values for the device itself will be under this hostname subdirectory, with the format <datasource>.<datapoint>.rrd; for example:

```
$ZENHOME/perf/Devices/zen241.class.example.org/laLoadInt1_laLoadInt1.rrd
$ZENHOME/perf/Devices/bino.skills-1st.co.uk/procs_linuxNum.rrd
```

If the object class of the device has contained components, such as *os*, which itself contains *filesystems* objects and *interfaces* objects, then the directory hierarchy under the hostname is extended to reflect and store the component data. Thus, interface information for the interface called eth1 on the device bino.skills-1st.co.uk would be stored in *$ZENHOME/perf/Devices/ bino.skills-1st.co.uk/os/interfaces/eth1* and would include datafiles such as *ifInOctets_ifInOctets.rrd*.

Note that a template must actually be **bound** to a device or device class before data collection will be effected (component templates are bound automatically).

Configuration data is collected by the **zenmodeler** daemon, each device or device class having been configured for one or more **modeler plugins**. The standard modeler plugins include SNMP, WMI and ssh as data collection protocols. Configuration data is stored in the Zope Object Database (ZODB).

Event data is stored in a MySQL relational database (that is installed and configured automatically when Zenoss is installed). The database has 6 tables:

- status
- history
- log
- detail
- heartbeat
- alert_status

The active events are in the status table whereas closed events are in the history table.

Events are generated and inserted into the database by various of the Zenoss daemons (such as **zenping**, **zenstatus** and **zenperfsnmp**). External events can also be captured and inserted from syslogs by the **zensyslog** daemon, from SNMP TRAPs by the **zentrap** daemon, and from Windows event logs by the **zeneventlog** daemon.

Figure 19 shows the directory structure for performance and configuration data.

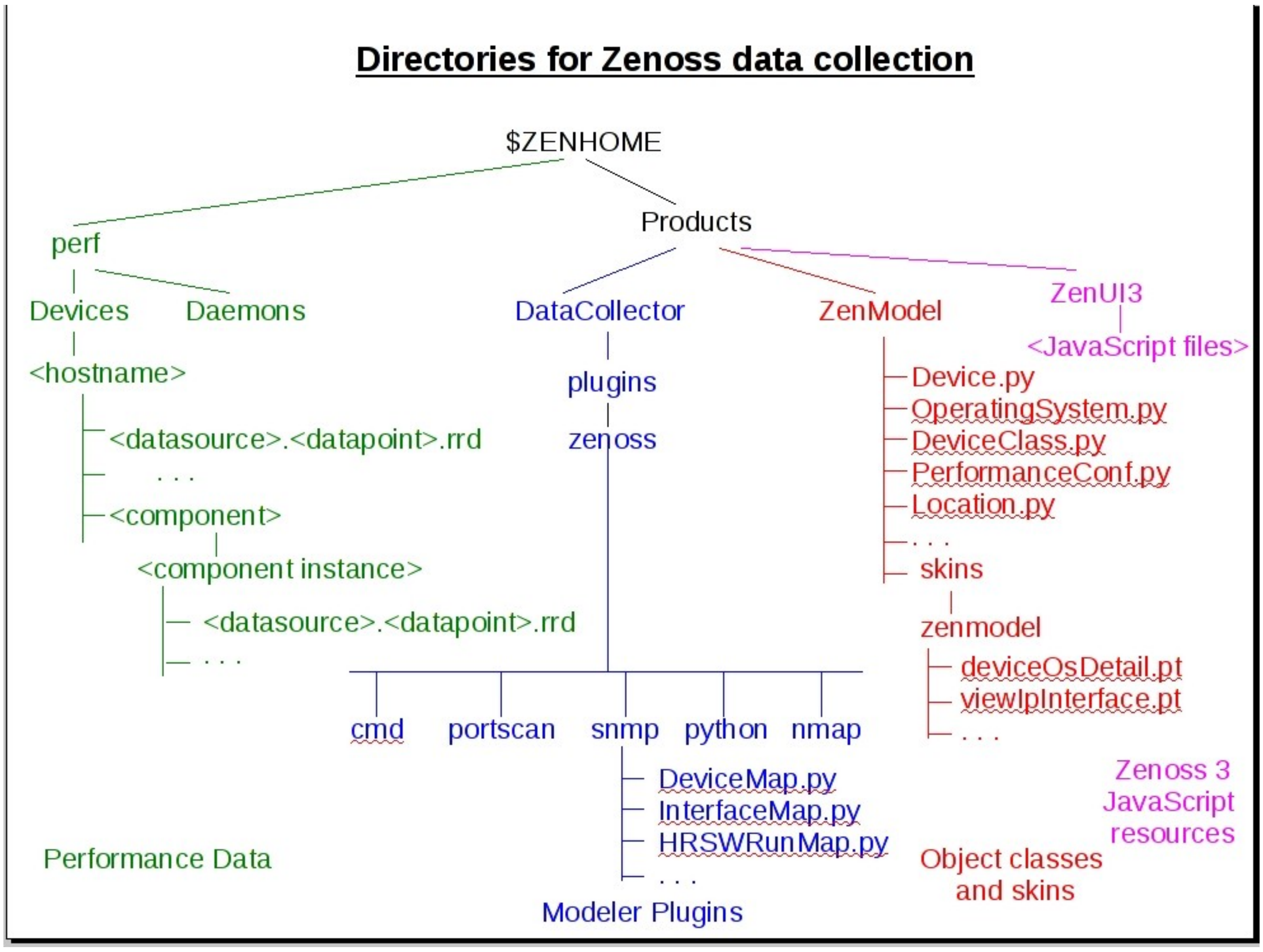
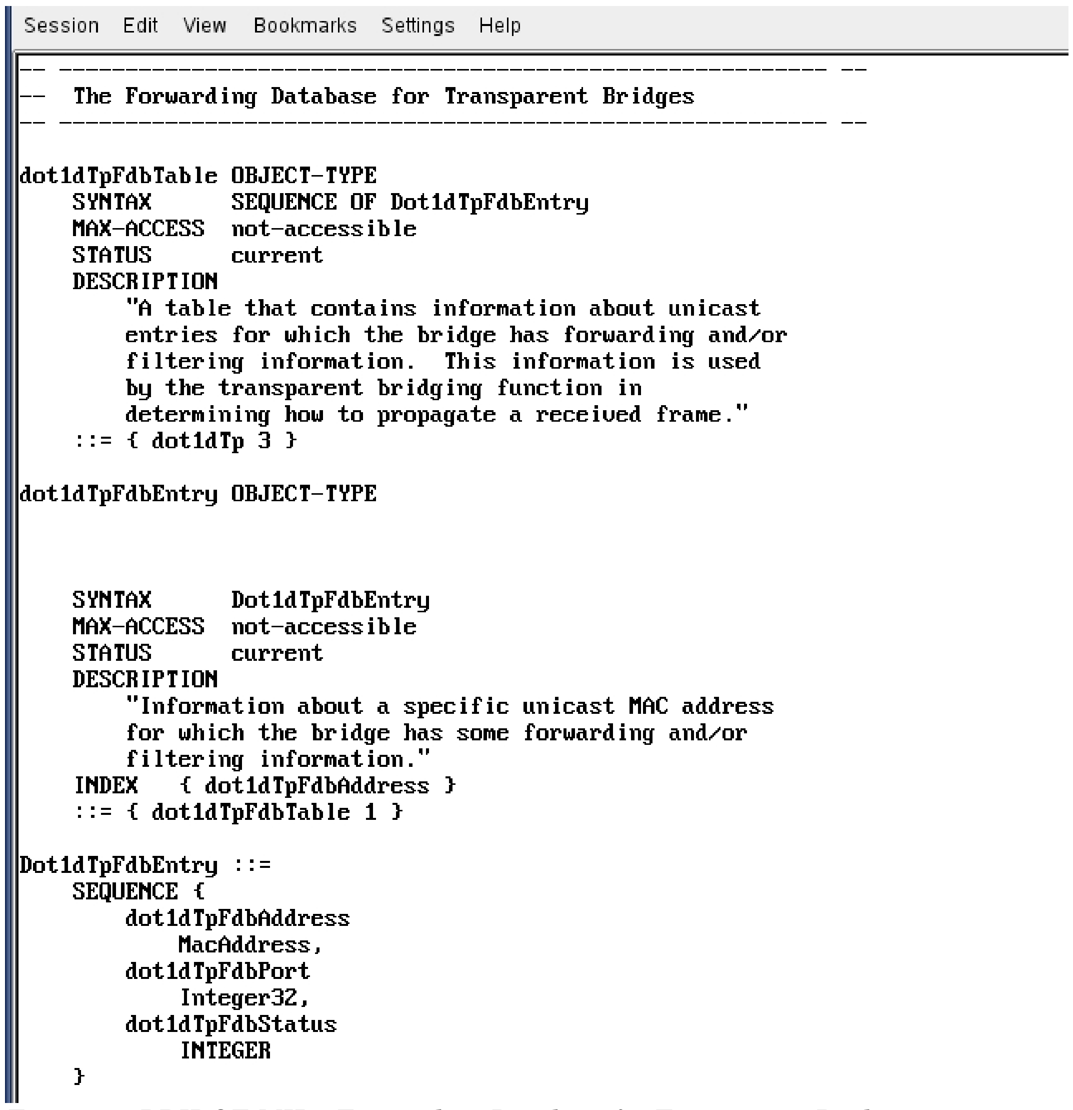


*Figure 19: Directory hierarchy for Zenoss data collection*

## 4.2 Requirements for the sample ZenPack

To illustrate the different elements of ZenPacks, a sample ZenPack will be created to get extra information from switch devices that support the BRIDGE MIB ( as defined by RFCs 1493 and 4188). The BRIDGE MIB provides information for each port on a switch, including the MAC address(es) that have been seen connected at the other end of the switch port; thus it is possible to build some ideas of layer 2 connectivity.

The main information that the BRIDGE MIB will supply to the ZenPack comes from the Forwarding Database for Transparent Bridges table (OID .1.3.6.1.2.1.17.4.3.1).



```
-- ---------------------------------------------------------- --
--  The Forwarding Database for Transparent Bridges
-- ---------------------------------------------------------- --

dot1dTpFdbTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF Dot1dTpFdbEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table that contains information about unicast
        entries for which the bridge has forwarding and/or
        filtering information.  This information is used
        by the transparent bridging function in
        determining how to propagate a received frame."
    ::= { dot1dTp 3 }

dot1dTpFdbEntry OBJECT-TYPE



    SYNTAX      Dot1dTpFdbEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Information about a specific unicast MAC address
        for which the bridge has some forwarding and/or
        filtering information."
    INDEX   { dot1dTpFdbAddress }
    ::= { dot1dTpFdbTable 1 }

Dot1dTpFdbEntry ::=
    SEQUENCE {
        dot1dTpFdbAddress
            MacAddress,
        dot1dTpFdbPort
            Integer32,
        dot1dTpFdbStatus
            INTEGER
    }
```

*Figure 20: BRIDGE MIB - Forwarding Database for Transparent Bridges section*

The three leaf-node OIDs are:

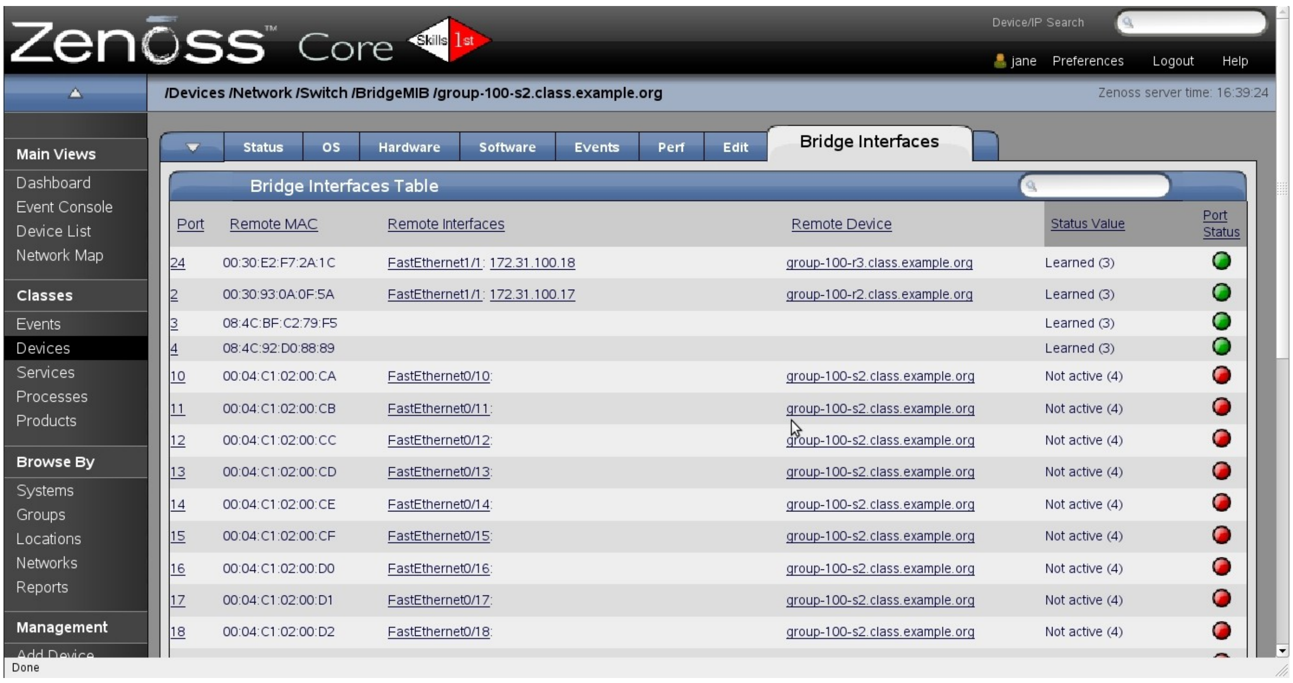
- dot1dTpFdbAddress of type MacAddress
- dot1dTpFdbPort of type Integer32
- dot1dTpFdbStatus – an enumerated INTEGER type where :
    - other(1),
    - invalid(2),
    - learned(3),
    - self(4),
    - mgmt(5)

For this ZenPack sample, only ports that have a status of "learned" (3) are going to be considered as active (ie. traffic has actively been seen going out of that port to a MAC address).

To make matters more confusing, the value supplied by the BRIDGE MIB for dot1dTpFdbPort for some switches,  does not match obvious port numbers.  For example, a Cisco Catalyst 2900 has 24 physical ports (actually labelled 1 – 24).  The BRIDGE MIB reports the first physical port as  dot1dTpFdbPort = 13, the second as 14, and so on.  To help a little with this confusion, the BRIDGE MIB provides the Generic Bridge Port Table (OID .1.3.6.1.2.1.17.1.4.1) whose first two leaf-node OIDs are:

- dot1dBasePort - "The port number of the port for which this entry contains bridge management information" – ie. the same port number as reported by dot1dTpFdbPort
- dot1dBasePortIfIndex - "The value of the instance of the ifIndex object, defined in IF-MIB, for the interface corresponding to this port."  In other words, this value provides a cross-reference between BRIDE MIB port references and their interfaces reported by the standard MIB-2 interface table.  For example, the port that is physically labelled 1, reports dot1dTpFdbPort=13, dot1dBasePortIfIndex=2 and information from the MIB-2 interface table for this port reports ifIndex=2 with the corresponding ifDescr="FastEthernet0/1" - I do hope that's clear!

So, to build any semblance of a layer 2 topology, we need to coordinate several pieces of information from the BRIDGE MIB and from MIB-2.  The target is to be able to display information as shown in Figure 21 for a Zenoss 2 system.



/Devices /Network /Switch /BridgeMIB /group-100-s2.class.example.org

Zenoss server time: 16:39:24

Bridge Interfaces Table

| Port | Remote MAC | Remote Interfaces | Remote Device | Status Value | Port Status |
|---|---|---|---|---|---|
| 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | Learned (3) | 🟢 |
| 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | Learned (3) | 🟢 |
| 3 | 08:4C:BF:C2:79:F5 | | | Learned (3) | 🟢 |
| 4 | 08:4C:92:D0:88:89 | | | Learned (3) | 🟢 |
| 10 | 00:04:C1:02:00:CA | FastEthernet0/10: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 11 | 00:04:C1:02:00:CB | FastEthernet0/11: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 12 | 00:04:C1:02:00:CC | FastEthernet0/12: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 13 | 00:04:C1:02:00:CD | FastEthernet0/13: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 14 | 00:04:C1:02:00:CE | FastEthernet0/14: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 15 | 00:04:C1:02:00:CF | FastEthernet0/15: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 16 | 00:04:C1:02:00:D0 | FastEthernet0/16: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 17 | 00:04:C1:02:00:D1 | FastEthernet0/17: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | Not active (4) | 🔴 |

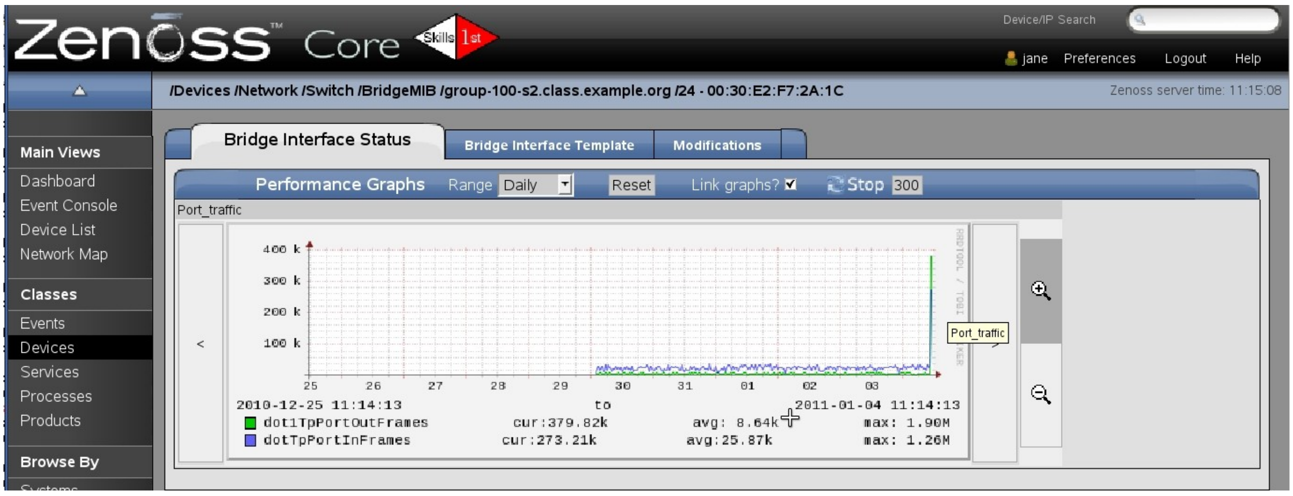
*Figure 21: The Bridge Interfaces Table for a Catalyst 2900 displayed by a Zenoss 2 system*

This screenshot shows four active MAC addresses two of which Zenoss is unable to provide remote information for (other than the remote MAC itself); this is because the remote device has not been discovered by Zenoss. Ports 2 and 24 are connected to devices already discovered so the remote interface description and IP address have been supplied out of the ZODB database, along with the hostname of the remote device. The rest of the ports are actually not connected so show a Port Status that is **not** 3.

In addition to getting port information from the BRIDGE MIB, it can also deliver its base bridge address as OID .1.3.6.1.2.1.17.1.1.0 and the total number of ports on the switch as OID .1.3.6.1.2.1.17.1.2.0. These values will be collected and displayed on the Status tab of a switch.

As shown in Figure 21, port information will be displayed in a new tab that will automatically be created for devices that support the BRIDGE MIB.

In addition to showing a table of ports, clicking on the *Port* link will display performance information for that port as shown in Figure 22.

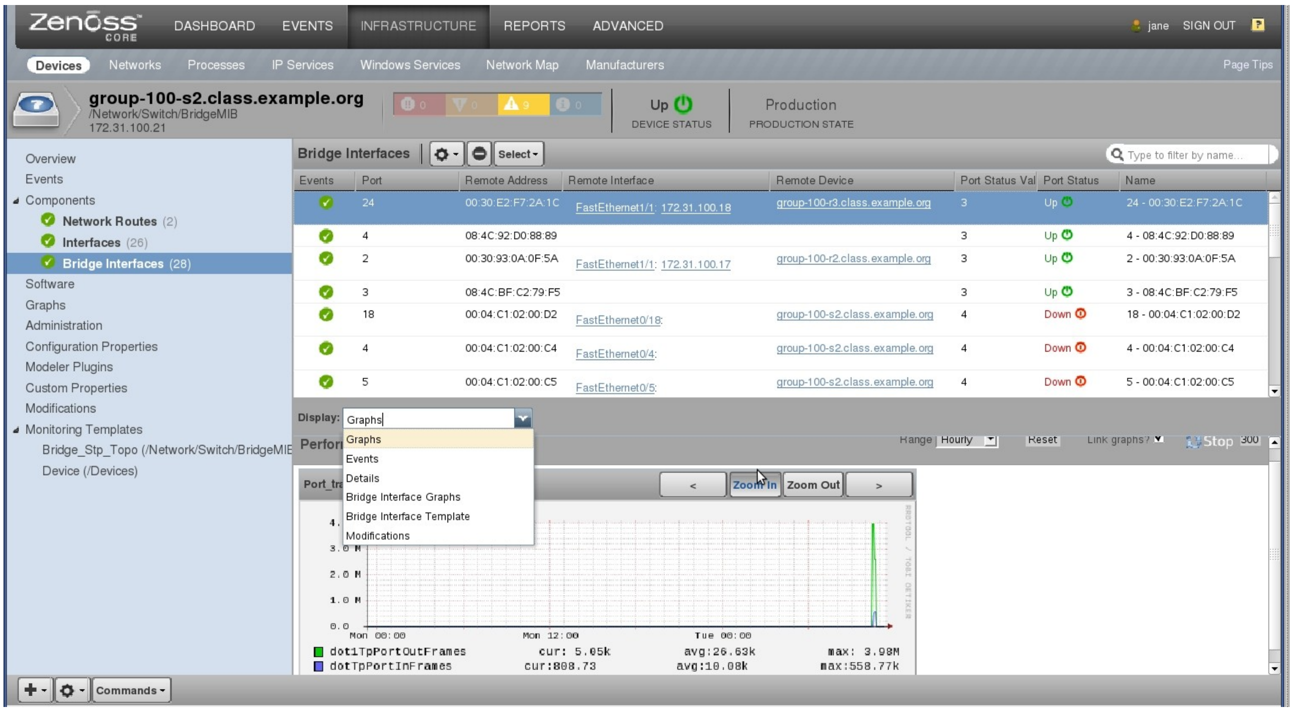


```
2010-12-25 11:14:13                to              2011-01-04 11:14:13
🟩 dot1TpPortOutFrames      cur:379.82k       avg: 8.64k        max: 1.90M
🟦 dotTpPortInFrames        cur:273.21k       avg:25.87k        max: 1.26M
```

*Figure 22: Performance graph for a selected switch port*

The template that delivers switch port performance information can have whatever datapoints you wish to configure but the template **name** must match the object class of the device component (BridgeInterface in this case) – more of this later.

In Zenoss 3, there is a slight design change. The Bridge Interfaces will be displayed as a component of a device, rather than being displayed as a separate element. The equivalent in Zenoss 2 would have been to extend the tables on the *OS* tab to include all the port information but this could have resulted in a huge OS page. Note that this is only a change in display technique – the fundamental device and component definitions do not change. Since Zenoss 3 provides the component submenu for a device, we will capitalise on it. Hence bridge interfaces will appear as shown in Figure 23.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Up DEVICE STATUS
Production PRODUCTION STATE

Bridge Interfaces

| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | Name |
|---|---|---|---|---|---|---|---|
| ✓ | 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | 3 | Up | 24 - 00:30:E2:F7:2A:1C |
| ✓ | 4 | 08:4C:92:D0:88:89 | | | 3 | Up | 4 - 08:4C:92:D0:88:89 |
| ✓ | 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | 3 | Up | 2 - 00:30:93:0A:0F:5A |
| ✓ | 3 | 08:4C:BF:C2:79:F5 | | | 3 | Up | 3 - 08:4C:BF:C2:79:F5 |
| ✓ | 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | 4 | Down | 18 - 00:04:C1:02:00:D2 |
| ✓ | 4 | 00:04:C1:02:00:C4 | FastEthernet0/4: | group-100-s2.class.example.org | 4 | Down | 4 - 00:04:C1:02:00:C4 |
| ✓ | 5 | 00:04:C1:02:00:C5 | FastEthernet0/5: | group-100-s2.class.example.org | 4 | Down | 5 - 00:04:C1:02:00:C5 |

```
🟩 dot1TpPortOutFrames     cur: 5.05k     avg:26.63k     max: 3.98M
▨ dotTpPortInFrames        cur:808.73     avg:10.08k     max:558.77k
```

*Figure 23: The Bridge Interfaces Table for a Catalyst 2900 displayed by a Zenoss 3 system*

The Graphs, Template and Modifications history that were represented in separate tabs in Figure 22, are now shown in the dropdown *Display* menu in the bottom half of the component panel.

## 4.3 Creating the sample ZenPack

It is essential to plan out the pieces of code required for a ZenPack and clearly document the names that will be used as many elements are referenced in other elements. Note that all names are case-sensitive.

### 4.3.1 Elements required and their names

This ZenPack is for devices that support the bridge MIB and it is created by Skills 1st, so the name of the ZenPack will be:

- **ZenPacks.skills1st.bridge**

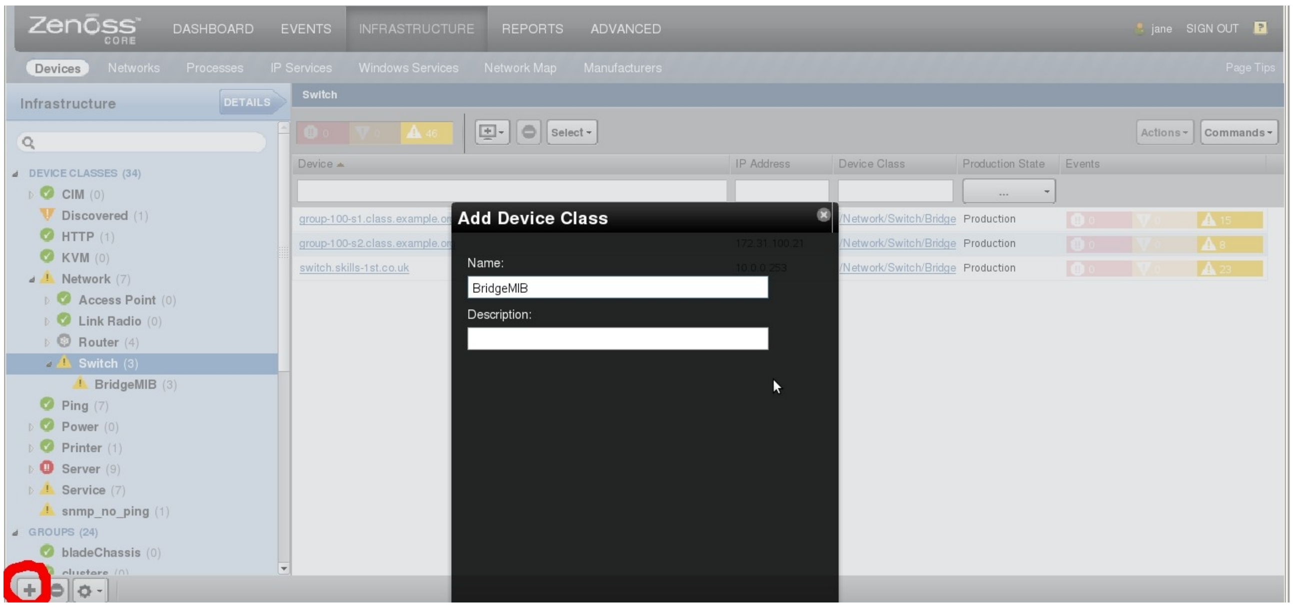
This means that a directory hierarchy will automatically be created under ZenPacks.skills1st.bridge:

- **ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge**

This directory will be referred to as the base directory of the ZenPack throughout this section, as it contains the object class files, the modeler directory and the skins directory. It also contains the resources directory for Zenoss 3.

A new device class will be used for Bridge MIB devices which is a subclass of the standard Switch device class. The device class is created through the GUI, simply by navigating to *Devices -> Network -> Switch* and using the "+" symbol at the bottom of the Zenoss 3 left-hand menu to add a sub device class. The new device class will be:

- **BridgeMIB**



Add Device Class

Name: BridgeMIB

Description:

*Figure 24: Creating a device class for BridgeMIB as a subclass of /Devices/Network/Switch*

(In Zenoss 2, use the table dropdown menu from *Devices* -> *Network* -> *Switch* to add a new organizer).

Note that this is a **device** class. It is not the **object** class file that specifies what makes such a device unique (relating the device class with the object class file will be discussed later).

The solution actually needs two new object class files; one for the device itself (**BridgeDevice**) and one to represent an interface on a bridge device (**BridgeInterface**). These files contain Python code and must exist in the base directory of the ZenPack. The name of the file should reflect the name of the object class that is being defined; thus:

- **BridgeDevice.py** contains the line **class BridgeDevice(Device):**
- **BridgeInterface.py** contains the line

  **class BridgeInterface(DeviceComponent, ManagedEntity):**

Within these object class files, the relationships between BridgeDevice and BridgeInterface will be specified (refer back to section 4.1.3 for information on relationships). Each relationship also has a name, distinct from the object class name so:

- A BridgeDevice object will have a relationship called **BridgeInt** defining a ToManyCont relationship with a BridgeInterface object (ie a BridgeDevice may contain many BridgeInterfaces).

- A BridgeInterface object will have a relationship called **BridgeDev** defining a ToOne relationship with a BridgeDevice object (ie. a BridgeInterface is associated with only one BridgeDevice).

Note that, by convention, the relationship name tends to reflect what is being related **to**.

Also note that some ZenPacks (especially older ones) define relationships in the __init__.py file of the base directory of the ZenPack. This procedure is also alluded to in the Zenoss Developer's Guide 2.4. My understanding, with recent versions of code, is that there is no requirement to modify any of the automatically-created __init__.py files if relationships are specified in object class files, as shown here.

Having created new object class files, modeler plugin code is required to populate the fields of these objects so the *modeler/plugins* directory under the ZenPack base directory contains:

- **BridgeDeviceMib.py**
- **BridgeInterfaceMib.py**

These files have Python code that use the standard SnmpPlugin collector to gather relevant SNMP data for the new objects. As discussed in section 4.1.1, modeler plugins are assigned to devices or device classes using the GUI with the *Modeler Plugins* left-hand menu (*More -> Collector Plugins* for Zenoss 2).

The final elements required are the web pages to show information about the new objects – these are held in the *skins/ZenPacks.skills1st.bridge* directory under the ZenPack base directory and have a *.pt* extension:

- **BridgeDeviceDetail.pt**
- **viewBridgeInterface.pt**

Zenoss 3 requires a JavaScript file to extend the details of the new component menu layout with Bridge Interface information. It is located under the *resources* directory:

- **bridge.js**

### 4.3.2 SNMP data required

Fundamentally, a protocol is necessary to gather both configuration and performance data. This ZenPack uses SNMP for both. It is always advisable to check that devices do respond to SNMP using a basic (non-Zenoss) SNMP command utility. The format of the command depends on the version of SNMP for which the device is configured. Here are examples for SNMP versions 1, 2 and 3, using the net-snmp utility, to walk the SNMP MIB tree from the BRIDGE MIB Forwarding Table (.1.3.6.1.2.1.17.4.3.1) for a device called switch (the hostname can be anything you can ping so potentially short hostnames will work just as well as fully-qualified Domain Names). SNMP versions 1 and 2c have a community name of *fraclmye* configured for use with the

Zenoss server. The SNMP V3 version uses *MD5* authentication, passphrase *fraclmyea*, and user *jane2*.

- `snmpwalk -v 1 -c fraclmye switch .1.3.6.1.2.1.17.4.3.1`
- `snmpwalk -v 2c -c fraclmye switch .1.3.6.1.2.1.17.4.3.1`
- `snmpwalk -v 3 - a MD5 -A fraclmyea -l authNoPriv -u jane2 switch .1.3.6.1.2.1.17.4.3.1`

Once basic SNMP communication is established, make sure that Zenoss device classes and/or devices have the correct SNMP parameters configured in their zProperties page.

The ZenPack will need two sets of table data from the BRIDGE MIB and two scalar values:

| | |
|---|---|
| .1.3.6.1.2.1.7.4.3.1 (dot1dTpFdbEntry) | .1 (RemoteAddress)<br>.2 (Port)<br>.3 (PortStatus) |
| .1.3.6.1.2.1.7.1.4.1 (dot1dBasePortEntry) | .1 (BasePort)<br>.2 (BasePortifIndex) |

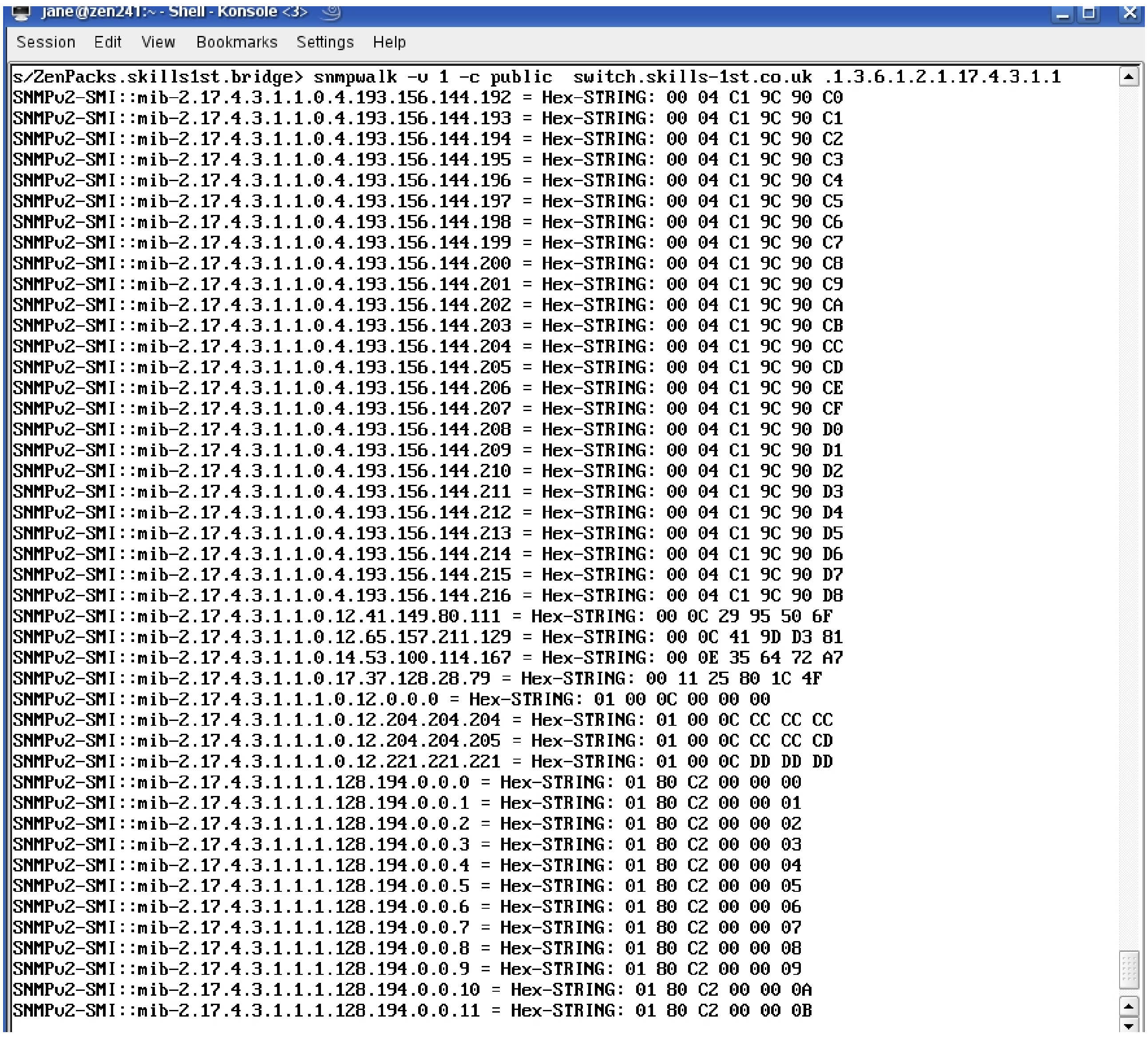
*Table 4.1.: Table entries from the BRIDGE MIB for each port of a switch*

| | |
|---|---|
| .1.3.6.1.2.1.17.1.1.0 | dot1dBaseBridgeAddress |
| .1.3.6.1.2.1.17.1.2.0 | dot1dBaseNumPorts |

*Table 4.2.: Scalar entries from the BRIDGE MIB*

Note that the *.0* is required on the end for the scalar MIB values.

Some of the values returned are shown in the following screenshots:


jane@zen241:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help

```
s/ZenPacks.skills1st.bridge> snmpwalk -v 1 -c public  switch.skills-1st.co.uk .1.3.6.1.2.1.17.4.3.1.1
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.192 = Hex-STRING: 00 04 C1 9C 90 C0
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.193 = Hex-STRING: 00 04 C1 9C 90 C1
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.194 = Hex-STRING: 00 04 C1 9C 90 C2
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.195 = Hex-STRING: 00 04 C1 9C 90 C3
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.196 = Hex-STRING: 00 04 C1 9C 90 C4
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.197 = Hex-STRING: 00 04 C1 9C 90 C5
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.198 = Hex-STRING: 00 04 C1 9C 90 C6
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.199 = Hex-STRING: 00 04 C1 9C 90 C7
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.200 = Hex-STRING: 00 04 C1 9C 90 C8
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.201 = Hex-STRING: 00 04 C1 9C 90 C9
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.202 = Hex-STRING: 00 04 C1 9C 90 CA
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.203 = Hex-STRING: 00 04 C1 9C 90 CB
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.204 = Hex-STRING: 00 04 C1 9C 90 CC
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.205 = Hex-STRING: 00 04 C1 9C 90 CD
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.206 = Hex-STRING: 00 04 C1 9C 90 CE
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.207 = Hex-STRING: 00 04 C1 9C 90 CF
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.208 = Hex-STRING: 00 04 C1 9C 90 D0
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.209 = Hex-STRING: 00 04 C1 9C 90 D1
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.210 = Hex-STRING: 00 04 C1 9C 90 D2
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.211 = Hex-STRING: 00 04 C1 9C 90 D3
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.212 = Hex-STRING: 00 04 C1 9C 90 D4
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.213 = Hex-STRING: 00 04 C1 9C 90 D5
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.214 = Hex-STRING: 00 04 C1 9C 90 D6
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.215 = Hex-STRING: 00 04 C1 9C 90 D7
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.216 = Hex-STRING: 00 04 C1 9C 90 D8
SNMPv2-SMI::mib-2.17.4.3.1.1.0.12.41.149.80.111 = Hex-STRING: 00 0C 29 95 50 6F
SNMPv2-SMI::mib-2.17.4.3.1.1.0.12.65.157.211.129 = Hex-STRING: 00 0C 41 9D D3 81
SNMPv2-SMI::mib-2.17.4.3.1.1.0.14.53.100.114.167 = Hex-STRING: 00 0E 35 64 72 A7
SNMPv2-SMI::mib-2.17.4.3.1.1.0.17.37.128.28.79 = Hex-STRING: 00 11 25 80 1C 4F
SNMPv2-SMI::mib-2.17.4.3.1.1.1.0.12.0.0.0 = Hex-STRING: 01 00 0C 00 00 00
SNMPv2-SMI::mib-2.17.4.3.1.1.1.0.12.204.204.204 = Hex-STRING: 01 00 0C CC CC CC
SNMPv2-SMI::mib-2.17.4.3.1.1.1.0.12.204.204.205 = Hex-STRING: 01 00 0C CC CC CD
SNMPv2-SMI::mib-2.17.4.3.1.1.1.0.12.221.221.221 = Hex-STRING: 01 00 0C DD DD DD
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.0 = Hex-STRING: 01 80 C2 00 00 00
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.1 = Hex-STRING: 01 80 C2 00 00 01
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.2 = Hex-STRING: 01 80 C2 00 00 02
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.3 = Hex-STRING: 01 80 C2 00 00 03
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.4 = Hex-STRING: 01 80 C2 00 00 04
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.5 = Hex-STRING: 01 80 C2 00 00 05
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.6 = Hex-STRING: 01 80 C2 00 00 06
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.7 = Hex-STRING: 01 80 C2 00 00 07
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.8 = Hex-STRING: 01 80 C2 00 00 08
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.9 = Hex-STRING: 01 80 C2 00 00 09
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.10 = Hex-STRING: 01 80 C2 00 00 0A
SNMPv2-SMI::mib-2.17.4.3.1.1.1.128.194.0.0.11 = Hex-STRING: 01 80 C2 00 00 0B
```

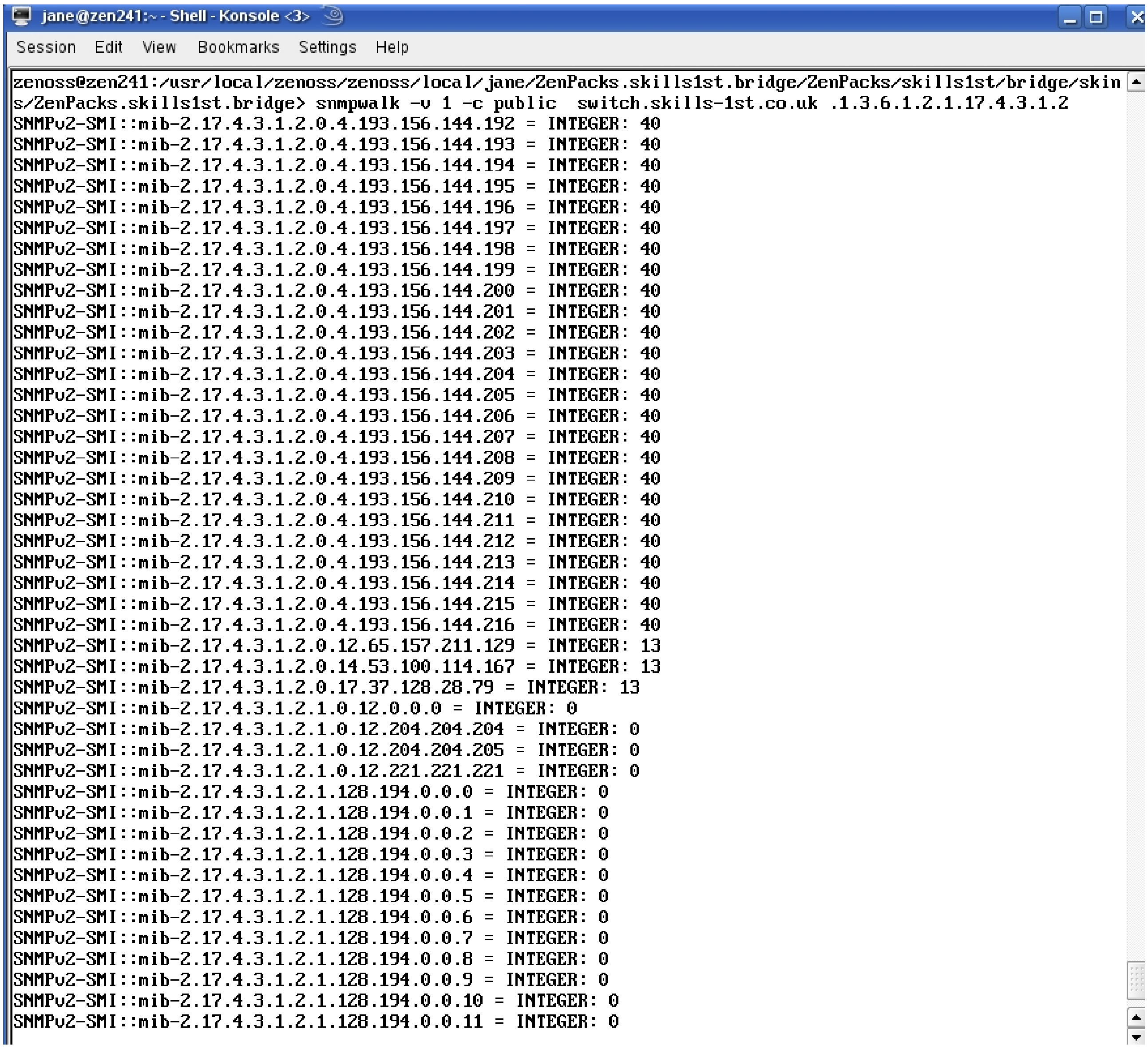
*Figure 25: Results from performing snmpwalk for the RemoteAddress values of the Port Forwarding table of the BRIDGE MIB*

Note that the OID index (the numbers after mib-2.17.4.3.1.1 represent the MAC address in decimal; thus in the first response of:

```
SNMPv2-SMI::mib-2.17.4.3.1.1.0.4.193.156.144.192 = Hex-STRING: 00 04 C1 9C 90 C0
```

the RemoteAddress MAC is *00 04 C1 9C 90 C0* and the index is *0.4.193.156.144.192* where:

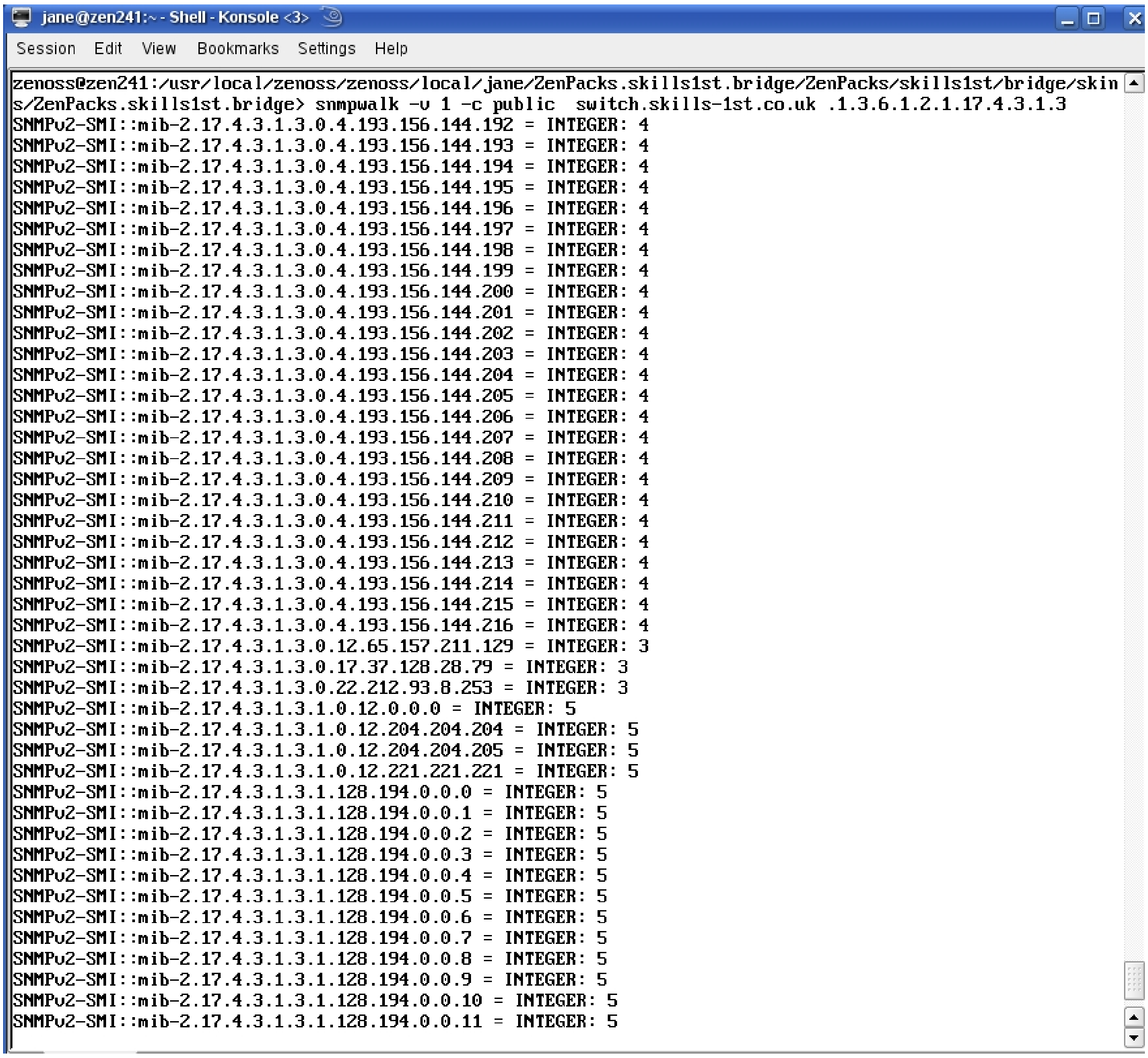
| **MAC Address** | **Index** | |
|---|---|---|
| 00 | 0 | |
| 04 | 4 | |
| C1 | 12 x 16 + 1 = 193 | |
| 9C | 9 x 16 + 12 = 156 | and so on |


jane@zen241:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help

```
zenoss@zen241:/usr/local/zenoss/zenoss/local/jane/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/skin
s/ZenPacks.skills1st.bridge> snmpwalk -v 1 -c public  switch.skills-1st.co.uk .1.3.6.1.2.1.17.4.3.1.2
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.192 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.193 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.194 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.195 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.196 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.197 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.198 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.199 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.200 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.201 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.202 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.203 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.204 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.205 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.206 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.207 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.208 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.209 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.210 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.211 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.212 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.213 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.214 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.215 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.4.193.156.144.216 = INTEGER: 40
SNMPv2-SMI::mib-2.17.4.3.1.2.0.12.65.157.211.129 = INTEGER: 13
SNMPv2-SMI::mib-2.17.4.3.1.2.0.14.53.100.114.167 = INTEGER: 13
SNMPv2-SMI::mib-2.17.4.3.1.2.0.17.37.128.28.79 = INTEGER: 13
SNMPv2-SMI::mib-2.17.4.3.1.2.1.0.12.0.0.0 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.0.12.204.204.204 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.0.12.204.204.205 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.0.12.221.221.221 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.0 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.1 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.2 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.3 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.4 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.5 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.6 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.7 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.8 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.9 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.10 = INTEGER: 0
SNMPv2-SMI::mib-2.17.4.3.1.2.1.128.194.0.0.11 = INTEGER: 0
```

*Figure 26: Results from performing snmpwalk for the Port values of the Port Forwarding table of the BRIDGE MIB*

The Port values are also indexed by the same representation of the MAC address in decimal. The only “real” values shown in Figure 26 are for port 13 (as only one port actually has anything connected to it). The other values of 0 and 40 are for internal and management addresses.


jane@zen241:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help

```
zenoss@zen241:/usr/local/zenoss/zenoss/local/jane/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/skin
s/ZenPacks.skills1st.bridge> snmpwalk -v 1 -c public  switch.skills-1st.co.uk .1.3.6.1.2.1.17.4.3.1.3
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.192 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.193 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.194 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.195 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.196 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.197 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.198 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.199 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.200 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.201 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.202 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.203 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.204 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.205 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.206 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.207 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.208 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.209 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.210 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.211 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.212 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.213 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.214 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.215 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.4.193.156.144.216 = INTEGER: 4
SNMPv2-SMI::mib-2.17.4.3.1.3.0.12.65.157.211.129 = INTEGER: 3
SNMPv2-SMI::mib-2.17.4.3.1.3.0.17.37.128.28.79 = INTEGER: 3
SNMPv2-SMI::mib-2.17.4.3.1.3.0.22.212.93.8.253 = INTEGER: 3
SNMPv2-SMI::mib-2.17.4.3.1.3.1.0.12.0.0.0 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.0.12.204.204.204 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.0.12.204.204.205 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.0.12.221.221.221 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.0 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.1 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.2 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.3 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.4 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.5 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.6 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.7 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.8 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.9 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.10 = INTEGER: 5
SNMPv2-SMI::mib-2.17.4.3.1.3.1.128.194.0.0.11 = INTEGER: 5
```

*Figure 27: Results from performing snmpwalk for the PortStatus values of the Port Forwarding table of the BRIDGE MIB*

The same indexing technique is used again. A PortStatus of 3 represents a “real” *learned* value. A value of 4 represents *self* addresses and a value of 5 represents *mgmt* addresses.

### 4.3.3 Creating the ZenPack

The ZenPack is created from the GUI as discussed in section 2.1. The name of the ZenPack is **ZenPacks.skills1st.bridge**. A Zenoss version dependency can also be imposed; for example, >=*3.0* .

Once the directory hierarchy is created, rather than working under $ZENHOME/ZenPacks, the whole ZenPack directory hierarchy is moved to $ZENHOME/local to prevent accidental deletion:

```
        cp -r $ZENHOME/ZenPacks/ZenPacks.skills1st.bridge $ZENHOME/local
```

The ZenPack is then “reinstalled” with the zenpack –link --install command:

```
        zenpack --link --install $ZENHOME/local/ZenPacks.skills1st.bridge
```

At this stage, the ZenPack can be modified either using Development mode (ie from GUI menus), or by modifying files in the directory hierarchy (Source mode); a combination of both is perfectly acceptable and both will follow the redirection link.

### 4.3.4 Adding elements to the ZenPack using Development mode

The first thing to do is to create the new device class, *BridgeMIB*, as a subclass of */Devices/Network/Switch*. This is simply achieved with the GUI as shown in Figure 24. Once created, this device class can be added to a ZenPack. In Zenoss 2 use the *Sub-Devices* drop-down *Add To ZenPack* menu – you will be prompted for the ZenPack to which it is to be added.

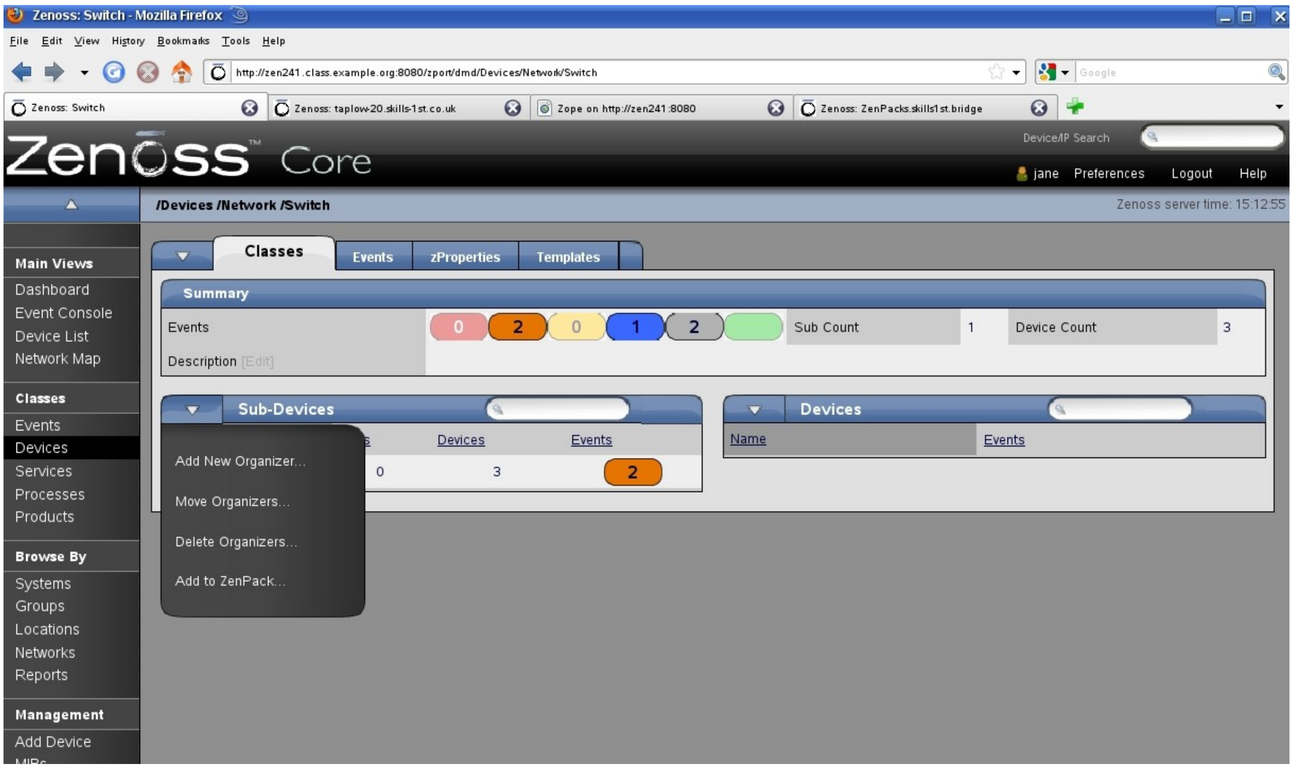
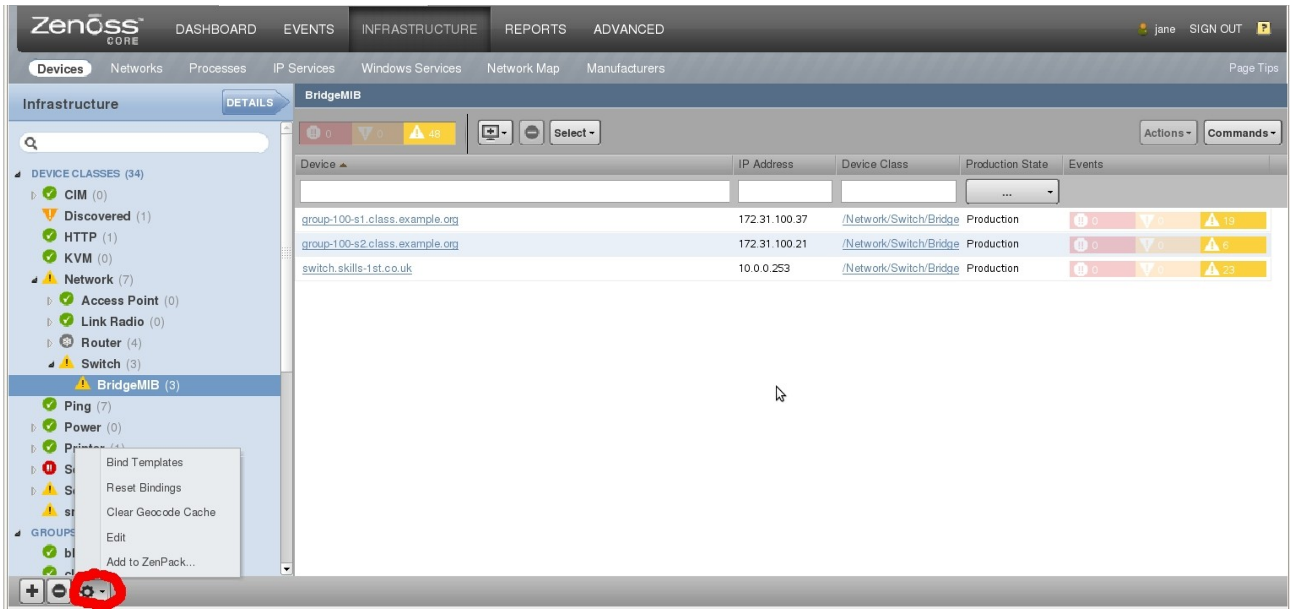


*Figure 28: Adding a device class to a ZenPack in Zenoss 2*

With Zenoss 3, GUI additions to ZenPacks are generally achieved from the “gear” *Action* icon and the *Add to ZenPack* option.

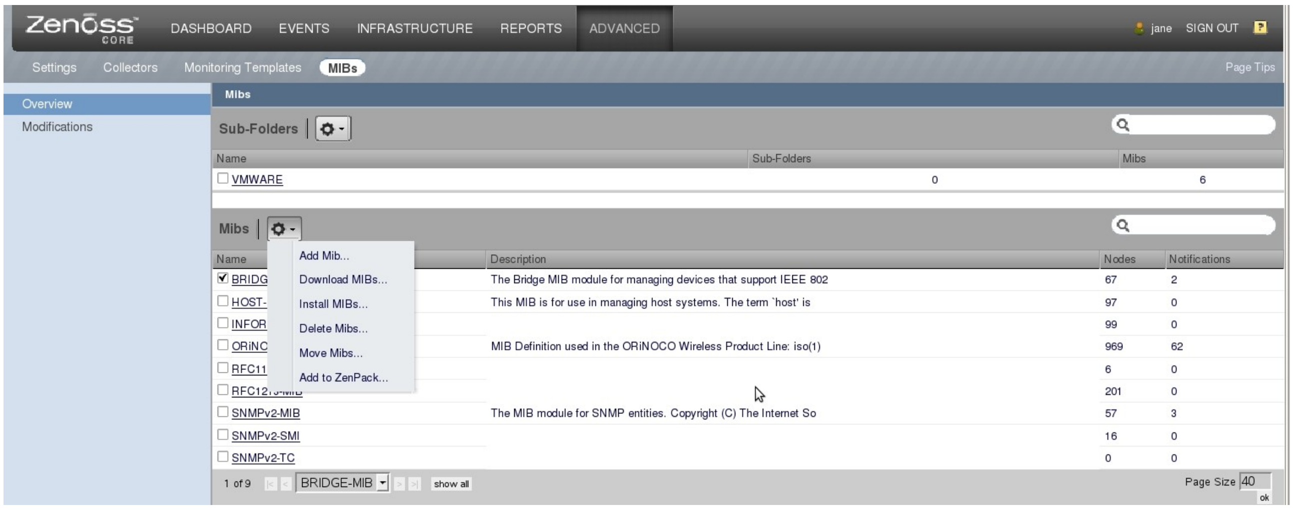


| Device | IP Address | Device Class | Production State | Events |
|---|---|---|---|---|
| group-100-s1.class.example.org | 172.31.100.37 | /Network/Switch/Bridge | Production | 0 0 19 |
| group-100-s2.class.example.org | 172.31.100.21 | /Network/Switch/Bridge | Production | 0 0 6 |
| switch.skills-1st.co.uk | 10.0.0.253 | /Network/Switch/Bridge | Production | 0 0 23 |

*Figure 29: Adding a device class to a ZenPack in Zenoss 3*

If the BridgeMIB device class is subsequently modified, it should be re-added in the same way, overwriting the previous version.

It is useful to have relevant MIBs loaded and included as part of any ZenPack. The BRIDGE MIB and the standard SNMP MIBs had already been loaded (using the left-hand *MIBs* menu in Zenoss 2 and the *ADVANCED -> MIBs* menu in Zenoss 3). To include these in a ZenPack, simply select the relevant MIBs and use the drop-down *Add to ZenPack* menu.



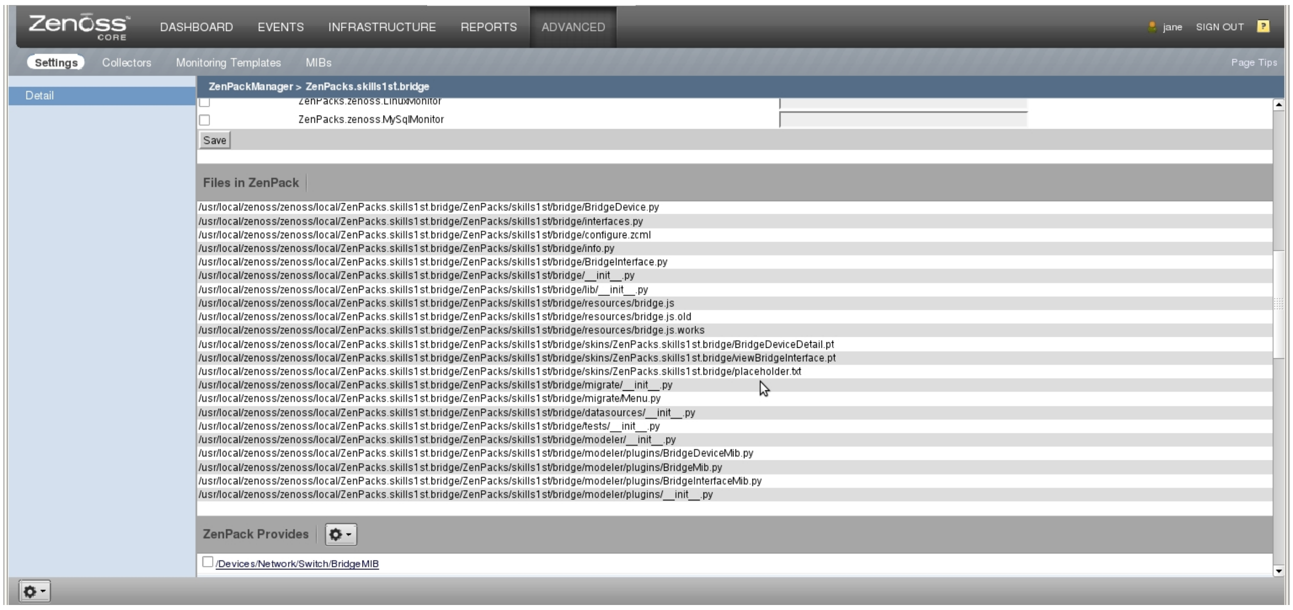
| Name | Sub-Folders | Mibs |
|---|---|---|
| VMWARE | 0 | 6 |

| Name | Description | Nodes | Notifications |
|---|---|---|---|
| ✓ BRIDG | The Bridge MIB module for managing devices that support IEEE 802 | 67 | 2 |
| HOST- | This MIB is for use in managing host systems. The term `host' is | 97 | 0 |
| INFOR | | 99 | 0 |
| ORiNC | MIB Definition used in the ORiNOCO Wireless Product Line: iso(1) | 969 | 62 |
| RFC11 | | 6 | 0 |
| RFC12 | | 201 | 0 |
| SNMPv2-MIB | The MIB module for SNMP entities. Copyright (C) The Internet So | 57 | 3 |
| SNMPv2-SMI | | 16 | 0 |
| SNMPv2-TC | | 0 | 0 |

1 of 9 BRIDGE-MIB show all Page Size 40

*Figure 30: Use the drop-down Mibs table menu to select Add to ZenPack*

The contents of a ZenPack can be seen at any stage by using the *ADVANCED -> Settings -> ZenPacks* option.



```
ZenPacks.zenoss.LinuxMonitor
ZenPacks.zenoss.MySqlMonitor
```

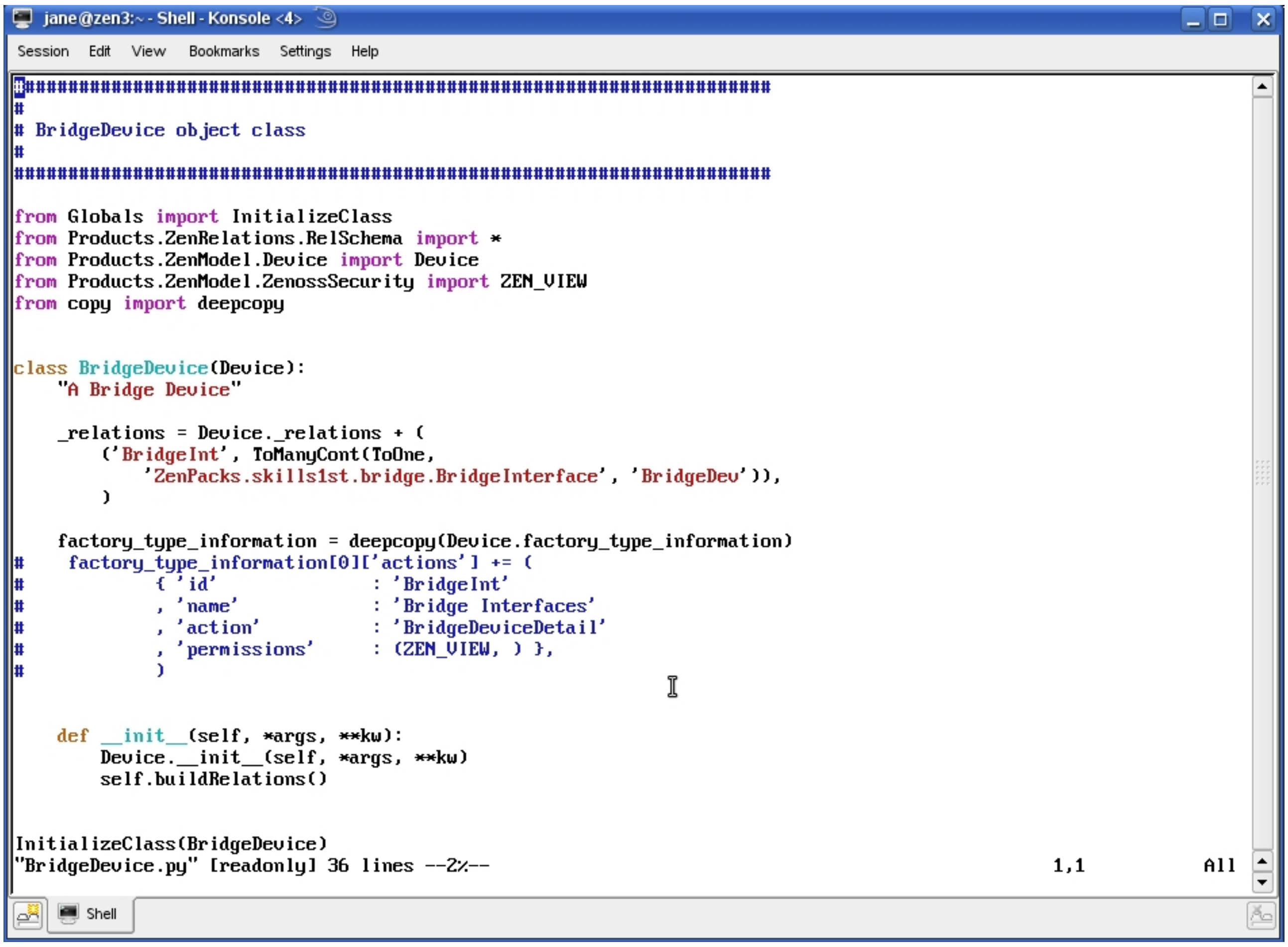
```
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/BridgeDevice.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/interfaces.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/configure.zcml
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/info.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/BridgeInterface.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/__init__.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/lib/__init__.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/resources/bridge.js
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/resources/bridge.js.old
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/resources/bridge.js.works
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/skins/ZenPacks.skills1st.bridge/BridgeDeviceDetail.pt
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/skins/ZenPacks.skills1st.bridge/viewBridgeInterface.pt
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/skins/ZenPacks.skills1st.bridge/placeholder.txt
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/migrate/__init__.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/migrate/Menu.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/datasources/__init__.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/tests/__init__.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/modeler/__init__.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/modeler/plugins/BridgeDeviceMib.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/modeler/plugins/BridgeMib.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/modeler/plugins/BridgeInterfaceMib.py
/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/modeler/plugins/__init__.py
```

```
/Devices/Network/Switch/BridgeMIB
```

*Figure 31: Inspecting the contents of the ZenPacks.skills1st.bridge ZenPack*

## 4.3.5 Creating the object class files

Two object class files are needed; one will represent the device itself (BridgeDevice) and one will represent an interface on the device (BridgeInterface). The two are linked by a matching pair of relationships. Both files must be in the ZenPack base directory.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
Shell

```
##########################################################################
#
# BridgeDevice object class
#
##########################################################################

from Globals import InitializeClass
from Products.ZenRelations.RelSchema import *
from Products.ZenModel.Device import Device
from Products.ZenModel.ZenossSecurity import ZEN_VIEW
from copy import deepcopy


class BridgeDevice(Device):
    "A Bridge Device"

    _relations = Device._relations + (
        ('BridgeInt', ToManyCont(ToOne,
            'ZenPacks.skills1st.bridge.BridgeInterface', 'BridgeDev')),
        )

    factory_type_information = deepcopy(Device.factory_type_information)
#    factory_type_information[0]['actions'] += (
#            { 'id'              : 'BridgeInt'
#            , 'name'            : 'Bridge Interfaces'
#            , 'action'          : 'BridgeDeviceDetail'
#            , 'permissions'     : (ZEN_VIEW, ) },
#            )


    def __init__(self, *args, **kw):
        Device.__init__(self, *args, **kw)
        self.buildRelations()


InitializeClass(BridgeDevice)
"BridgeDevice.py" [readonly] 36 lines --2%--                        1,1           All
```

Figure 32: BridgeDevice.py object class file in ZenPack base directory

This is a very simple object class file as it does not define any unique field attributes, only a relationship and a skins file – and it only needs the skins file with Zenoss 2 (Figure 32 shows the extra tab commented out for Zenoss 3).

The BridgeDevice class inherits from the base Device class:

```
class BridgeDevice(Device):
```

The relationship stanza adopts all existing relations for the base Device class and adds on a relationship called *BridgeInt* of type *ToManyCont*, with the device object class defined in *ZenPacks.skills1st.bridge.BridgeInterface* (which corresponds to the file under the ZenPack base directory called *BridgeInterface.py*).

```
_relations = Device._relations + (
     'BridgeInt', ToManyCont(ToOne,
        'ZenPacks.skills1st.bridge.BridgeInterface', 'BridgeDev')),
        )
```

The BridgeDevice object class will have all the standard menu options for the base Device class and, in Zenoss 2, will also have an extra tab whose id is *BridgeInt*; whose tab label will be *Bridge Interfaces*; and whose page layout will be specified by the file *BridgeDeviceDetail*.pt under the skins/ZenPacks.skills1st.bridge subdirectory of the

ZenPack base directory. Access permissions to use this tab is the standard-supplied *ZEN_VIEW*.

```
        factory_type_information = deepcopy(Device.factory_type_information)
        factory_type_information[0]['actions'] += (
                { 'id'              : 'BridgeInt'
                , 'name'            : 'Bridge Interfaces'
                , 'action'          : 'BridgeDeviceDetail'
                , 'permissions'     : (ZEN_VIEW, ) },
                )
```

Note that if these lines are left uncommented in Zenoss 3 then the left-hand menu for a BridgeMIB device will also have a separate *Bridge Interfaces* menu, in addition to the same information under the Component submenu.

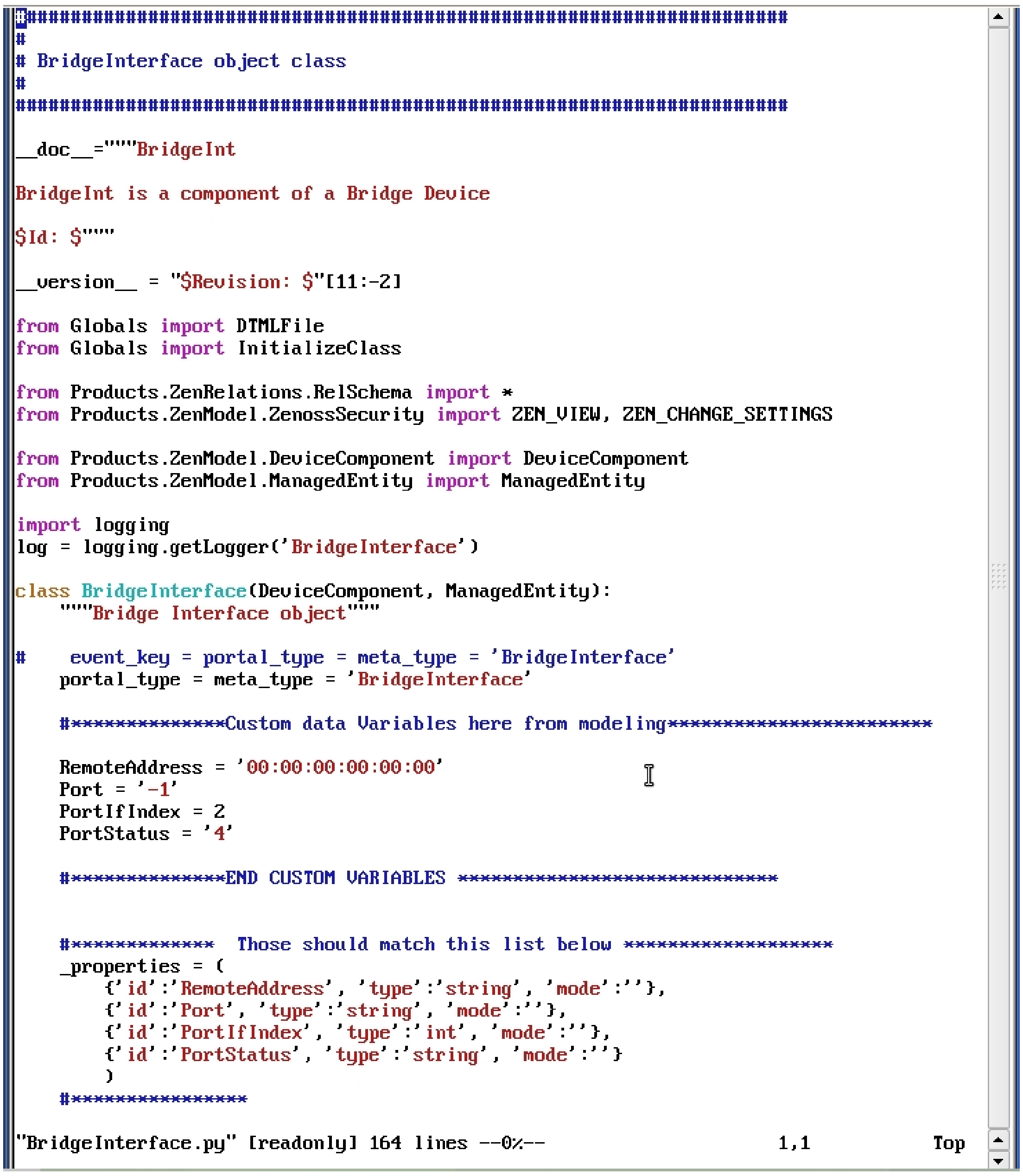
A Python function *__init__* is defined for the BridgeDevice object class which will initialize the object and create relationships.

```
    def __init__(self, *args, **kw):
            Device.__init__(self, *args, **kw)
            self.buildRelations()
```

The last line delivers the new object.

```
     InitializeClass(BridgeDevice)
```

The BridgeInterface.py object class file is more interesting as some unique fields are defined in addition to a relationship and a skins file. Several extra functions are also defined which will be used in the skins files.



```
##########################################################################
#
# BridgeInterface object class
#
##########################################################################

__doc__="""BridgeInt

BridgeInt is a component of a Bridge Device

$Id: $"""

__version__ = "$Revision: $"[11:-2]

from Globals import DTMLFile
from Globals import InitializeClass

from Products.ZenRelations.RelSchema import *
from Products.ZenModel.ZenossSecurity import ZEN_VIEW, ZEN_CHANGE_SETTINGS

from Products.ZenModel.DeviceComponent import DeviceComponent
from Products.ZenModel.ManagedEntity import ManagedEntity

import logging
log = logging.getLogger('BridgeInterface')

class BridgeInterface(DeviceComponent, ManagedEntity):
    """Bridge Interface object"""

#    event_key = portal_type = meta_type = 'BridgeInterface'
    portal_type = meta_type = 'BridgeInterface'

    #***************Custom data Variables here from modeling************************

    RemoteAddress = '00:00:00:00:00:00'
    Port = '-1'
    PortIfIndex = 2
    PortStatus = '4'

    #***************END CUSTOM VARIABLES ******************************


    #**************  Those should match this list below *******************
    _properties = (
        {'id':'RemoteAddress', 'type':'string', 'mode':''},
        {'id':'Port', 'type':'string', 'mode':''},
        {'id':'PortIfIndex', 'type':'int', 'mode':''},
        {'id':'PortStatus', 'type':'string', 'mode':''}
        )
    #*****************

"BridgeInterface.py" [readonly] 164 lines --0%--                    1,1           Top
```

*Figure 33: BridgeInterface.py object class file - first part with unique field definitions*

The BridgeInterface class inherits attributes from the DeviceComponent and ManagedEntity classes.

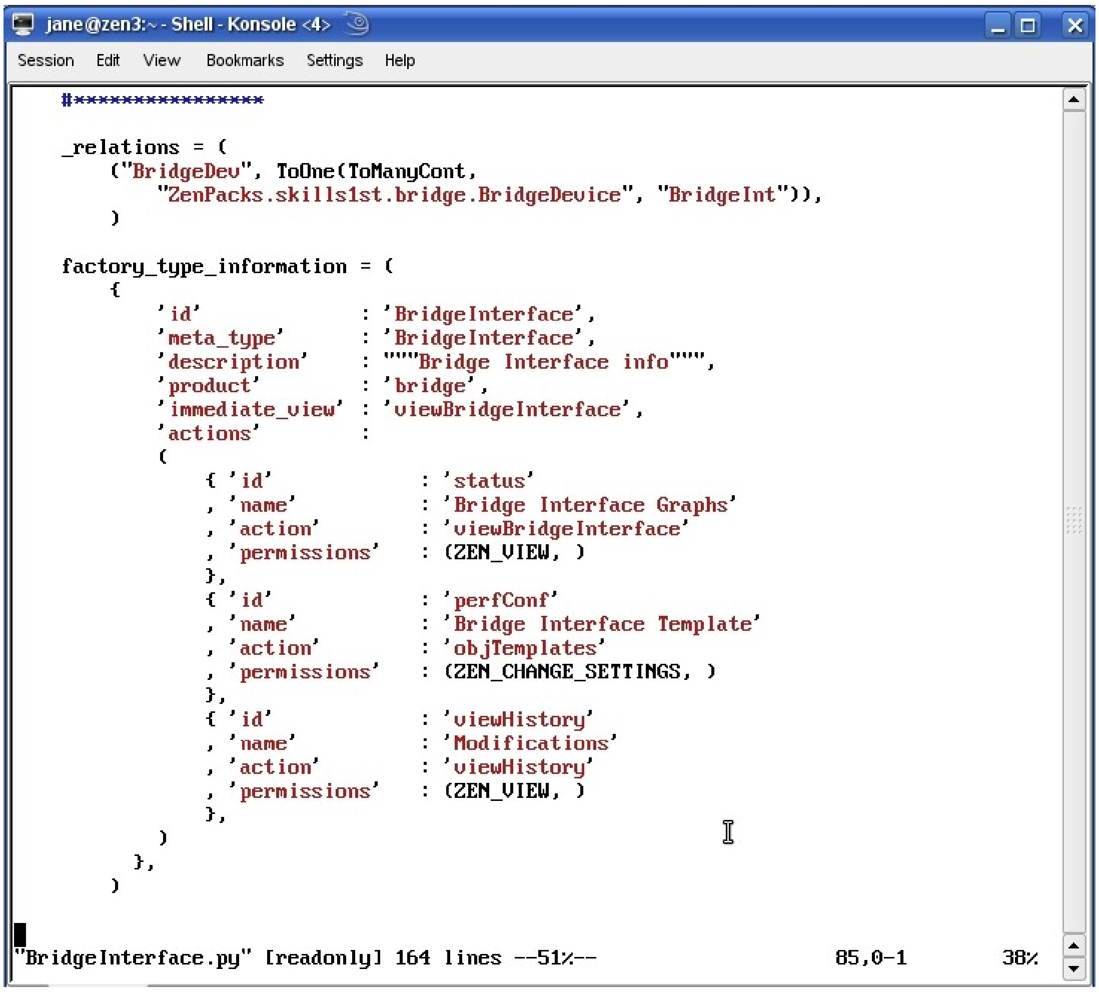
```
class BridgeInterface(DeviceComponent, ManagedEntity):
```

There are 4 unique fields defined for a BridgeInterface object:

- RemoteAddress
- Port
- PortIfIndex
- PortStatus

The data types (such as *string* or *int*) must be specified and the mode of read or write ('w') may be specified.

The middle part of BridgeInterface.py defines the relationship with BridgeDevice and three web pages associated with this object.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
    #*****************

    _relations = (
        ("BridgeDev", ToOne(ToManyCont,
            "ZenPacks.skills1st.bridge.BridgeDevice", "BridgeInt")),
        )

    factory_type_information = (
        {
            'id'             : 'BridgeInterface',
            'meta_type'      : 'BridgeInterface',
            'description'    : """Bridge Interface info""",
            'product'        : 'bridge',
            'immediate_view' : 'viewBridgeInterface',
            'actions'        :
            (
                { 'id'            : 'status'
                , 'name'          : 'Bridge Interface Graphs'
                , 'action'        : 'viewBridgeInterface'
                , 'permissions'   : (ZEN_VIEW, )
                },
                { 'id'            : 'perfConf'
                , 'name'          : 'Bridge Interface Template'
                , 'action'        : 'objTemplates'
                , 'permissions'   : (ZEN_CHANGE_SETTINGS, )
                },
                { 'id'            : 'viewHistory'
                , 'name'          : 'Modifications'
                , 'action'        : 'viewHistory'
                , 'permissions'   : (ZEN_VIEW, )
                },
            )
          },
        )

"BridgeInterface.py" [readonly] 164 lines --51%--           85,0-1        38%
```

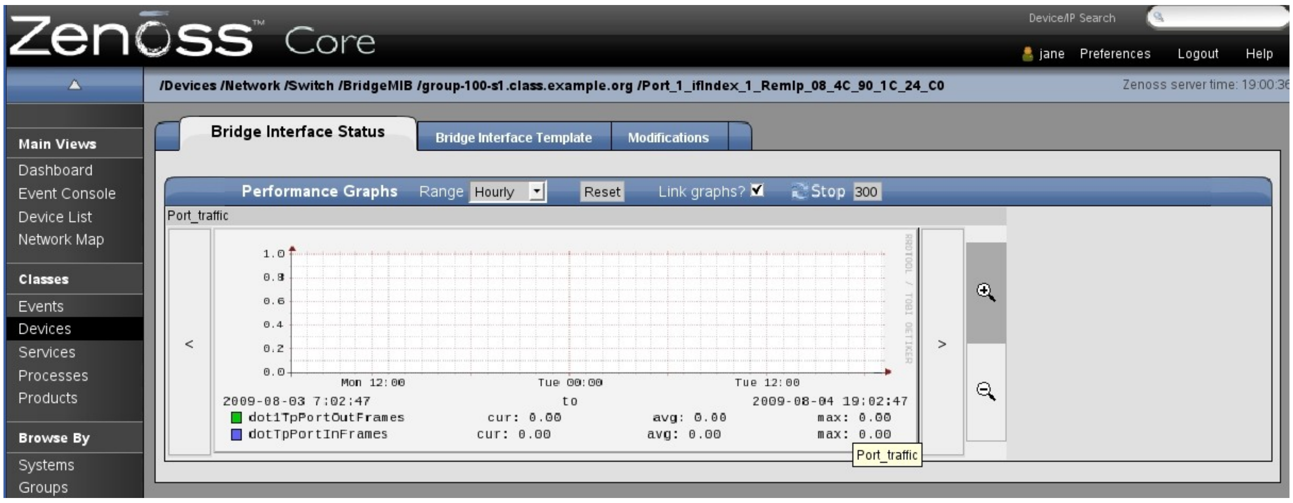
*Figure 34: BridgeInterface.py showing relations and web pages*

The first skins file that is referenced, *viewBridgeInterface*, is part of this ZenPack and thus is to be found in the skins/ZenPacks.skills1st.bridge subdirectory; the other two files, *objTemplates* and *viewHistory* are standard pages provided by Zenoss and these .pt files are found in $ZENHOME/Products/ZenModel/skins/zenmodel.

Note the product line in the factory_type_information:

```
    'product'          : 'bridge'
```

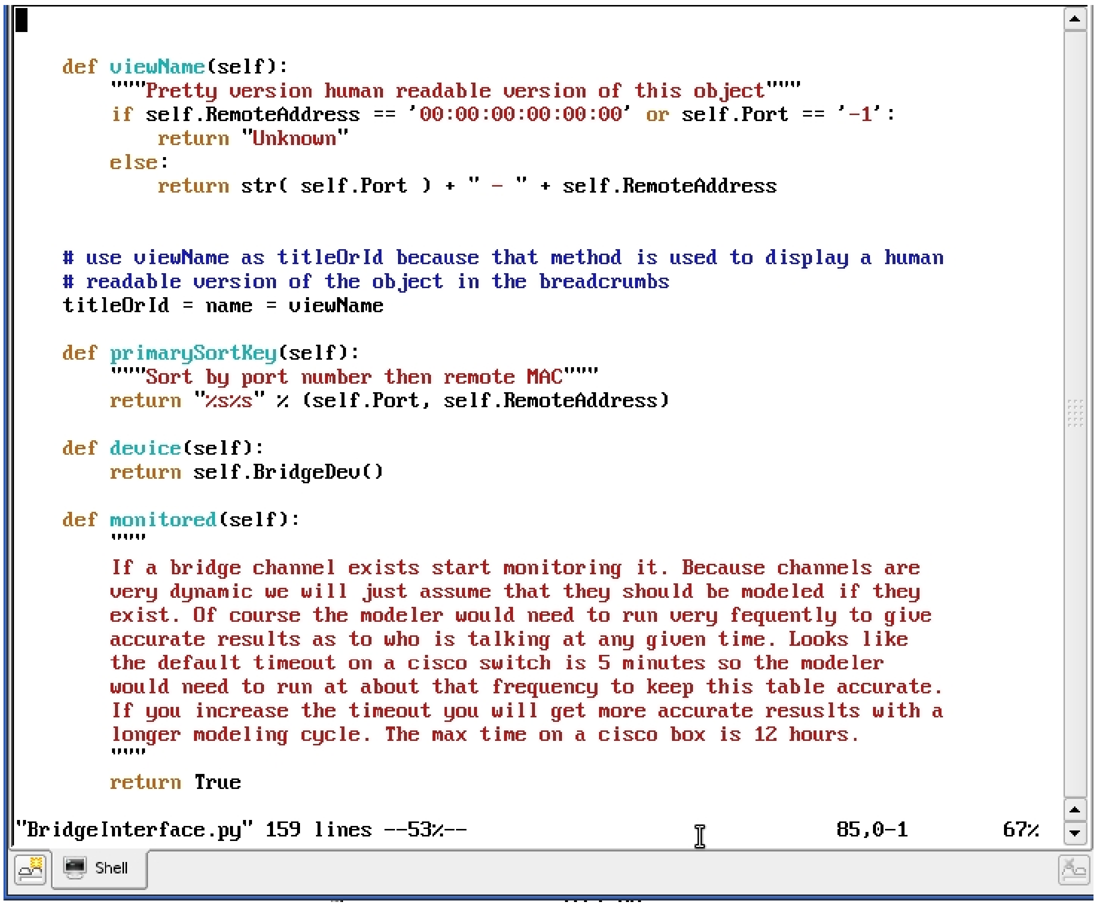
The value of *'bridge'* denotes the last part of the ZenPack name (and hence the directory hierarchy) ie. ZenPacks.skills1st.**bridge**. Unlike the BridgeDevice definitions of skins files, nothing is inherited from the base Device object. The resulting Zenoss 2 web page with its three tabs, can be seen in Figure 35.



```
2009-08-03 7:02:47              to              2009-08-04 19:02:47
■ dot1TpPortOutFrames     cur: 0.00     avg: 0.00     max: 0.00
■ dotTpPortInFrames       cur: 0.00     avg: 0.00     max: 0.00
```

Figure 35: The Zenoss 2 web page having drilled into the interface of a switch port - note the 3 tabs

An object class definition file can specify not only object attributes but also methods for the object; these are coded as function definitions in Python. The methods can then be used in skins files to augment the data that is displayed.



```
    def viewName(self):
        """Pretty version human readable version of this object"""
        if self.RemoteAddress == '00:00:00:00:00:00' or self.Port == '-1':
            return "Unknown"
        else:
            return str( self.Port ) + " - " + self.RemoteAddress


    # use viewName as titleOrId because that method is used to display a human
    # readable version of the object in the breadcrumbs
    titleOrId = name = viewName

    def primarySortKey(self):
        """Sort by port number then remote MAC"""
        return "%s%s" % (self.Port, self.RemoteAddress)

    def device(self):
        return self.BridgeDev()

    def monitored(self):
        """
        If a bridge channel exists start monitoring it. Because channels are
        very dynamic we will just assume that they should be modeled if they
        exist. Of course the modeler would need to run very fequently to give
        accurate results as to who is talking at any given time. Looks like
        the default timeout on a cisco switch is 5 minutes so the modeler
        would need to run at about that frequency to keep this table accurate.
        If you increase the timeout you will get more accurate resuslts with a
        longer modeling cycle. The max time on a cisco box is 12 hours.
        """
        return True
"BridgeInterface.py" 159 lines --53%--                    85,0-1        67%
```

Figure 36: BridgeInterface.py part 3 showing basic functions defined for this object class

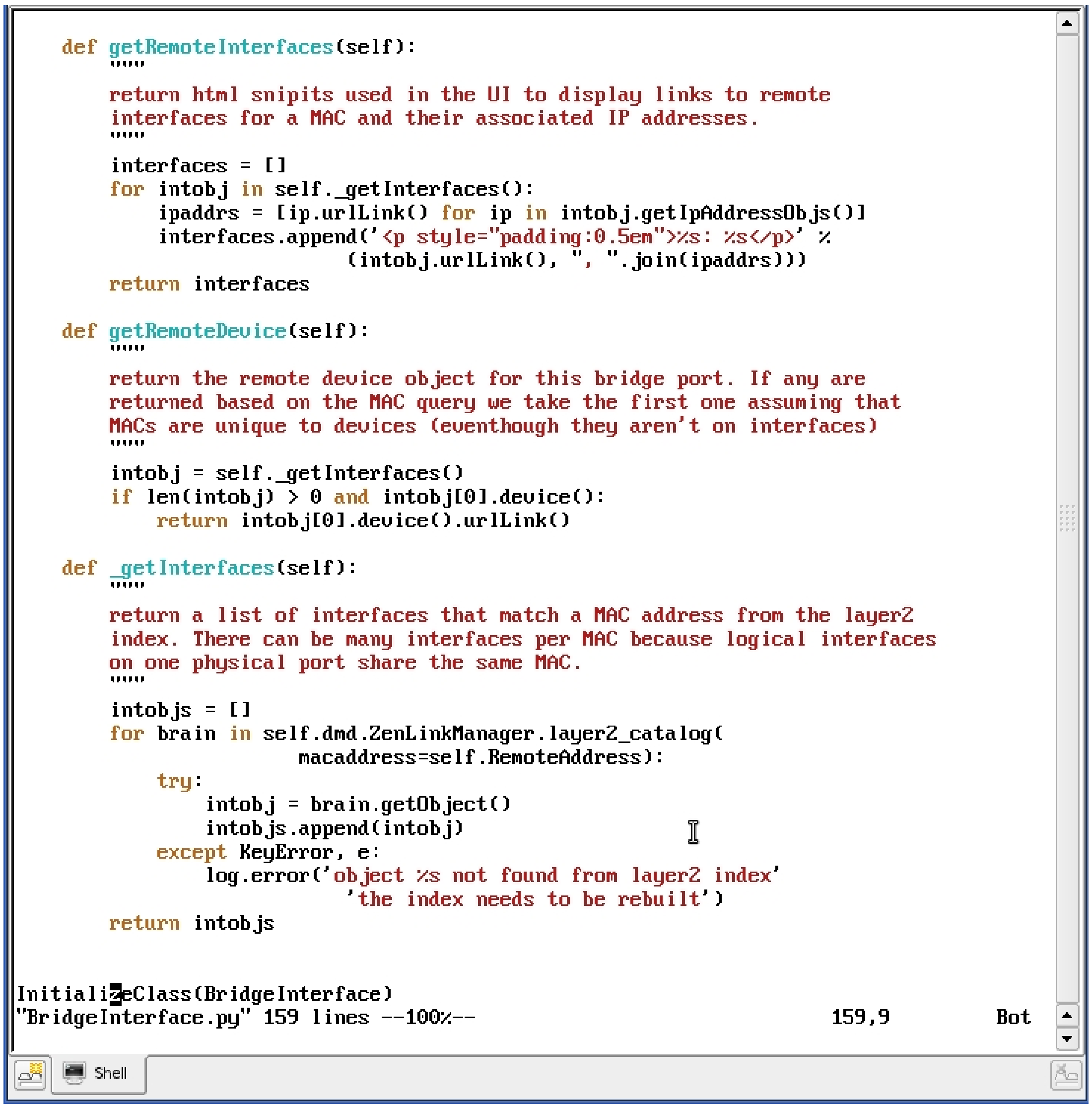
The first function, **viewName**, returns either the string “Unknown” or a string that concatenates the Port number with a “ - ” and the RemoteAddress. For example, *13 - 00:11:25:80:1C:4F* .

```
        def viewName(self):
            if self.RemoteAddress == '00:00:00:00:00:00' \
                or self.Port == '-1':
                    return "Unknown"
            else:
                    return str( self.Port ) + " - " + self.RemoteAddress
```

Zenoss 2.5.0 introduced a new device attribute of **titleOrId**. In Zenoss versions prior to 2.5, a single identifier ("id") was used to represent a device in the system and in the user interface. Beginning with 2.5.0, a separate "title" property, if specified, replaces the name of the device in the user interface. (The "id" property is retained as the internal, unique representation of the device.) This addition accommodates situations in which a unique identifier and a "friendly" name are needed for a device. It also allows devices to use a more descriptive or short name in the GUI rather than the more typical fully-qualified domain name. Thus, BridgeInterface.py has the following line after defining viewName:

```
    titleOrId = name = viewName
```

The remainder of BridgeInterface.py contains three method functions, two of which can be used in skins files.



```
    def getRemoteInterfaces(self):
        """
        return html snipits used in the UI to display links to remote
        interfaces for a MAC and their associated IP addresses.
        """
        interfaces = []
        for intobj in self._getInterfaces():
            ipaddrs = [ip.urlLink() for ip in intobj.getIpAddressObjs()]
            interfaces.append('<p style="padding:0.5em">%s: %s</p>' %
                        (intobj.urlLink(), ", ".join(ipaddrs)))
        return interfaces

    def getRemoteDevice(self):
        """
        return the remote device object for this bridge port. If any are
        returned based on the MAC query we take the first one assuming that
        MACs are unique to devices (eventhough they aren't on interfaces)
        """
        intobj = self._getInterfaces()
        if len(intobj) > 0 and intobj[0].device():
            return intobj[0].device().urlLink()

    def _getInterfaces(self):
        """
        return a list of interfaces that match a MAC address from the layer2
        index. There can be many interfaces per MAC because logical interfaces
        on one physical port share the same MAC.
        """
        intobjs = []
        for brain in self.dmd.ZenLinkManager.layer2_catalog(
                        macaddress=self.RemoteAddress):
            try:
                intobj = brain.getObject()
                intobjs.append(intobj)
            except KeyError, e:
                log.error('object %s not found from layer2 index'
                        'the index needs to be rebuilt')
        return intobjs


InitializeClass(BridgeInterface)
"BridgeInterface.py" 159 lines --100%--                        159,9         Bot
```

*Figure 37: Methods in BridgeInterface.py for use in skins files*

The private function *_getInterfaces* (underscore at the beginning of a variable or function name denotes a private object, by convention in Python) uses the MAC address delivered by the RemoteAddress field and then searches the Zope database for devices that have matching MAC address(es) – there may be more than one IP address associated with a MAC address.  A Python list of **interface objects** is returned by calling the standard Zenoss *getObject()* method for any matching MAC address.

```
    def _getInterfaces(self):
        """
        return a list of interfaces that match a MAC address from the layer2
        index. There can be many interfaces per MAC because logical interfaces
        on one physical port share the same MAC.
        """
        intobjs = []
        for brain in self.dmd.ZenLinkManager.layer2_catalog(
                        macaddress=self.RemoteAddress):
            try:
                intobj = brain.getObject()
                intobjs.append(intobj)
            except KeyError, e:
                log.error('object %s not found from layer2 index'
                            'the index needs to be rebuilt')
        return intobjs
```

This private method is called by both the other methods, *getRemoteInterfaces* and *getRemoteDevice*, which deliver links to matching IP interfaces and hostname, respectively.

```
    def getRemoteInterfaces(self):
        """
        return html snipits used in the UI to display links to remote
        interfaces for a MAC and their associated IP addresses.
        """
        interfaces = []
        for intobj in self._getInterfaces():
            ipaddrs = [ip.urlLink() for ip in intobj.getIpAddressObjs()]
            interfaces.append('<p style="padding:0.5em">%s: %s</p>' %
                            (intobj.urlLink(), ", ".join(ipaddrs)))
        return interfaces
```

The standard Zenoss *getIpAddressObjs()* method is called for each interface object to deliver an IP address object and then the urlLink variable for that IP address. getRemoteInterfaces returns a list of interfaces in the format:

```
<urlLink to remote interface object>: <urlLink to remote IP address>
```

The getRemoteDevice method simply returns the urlLink to the remote device.

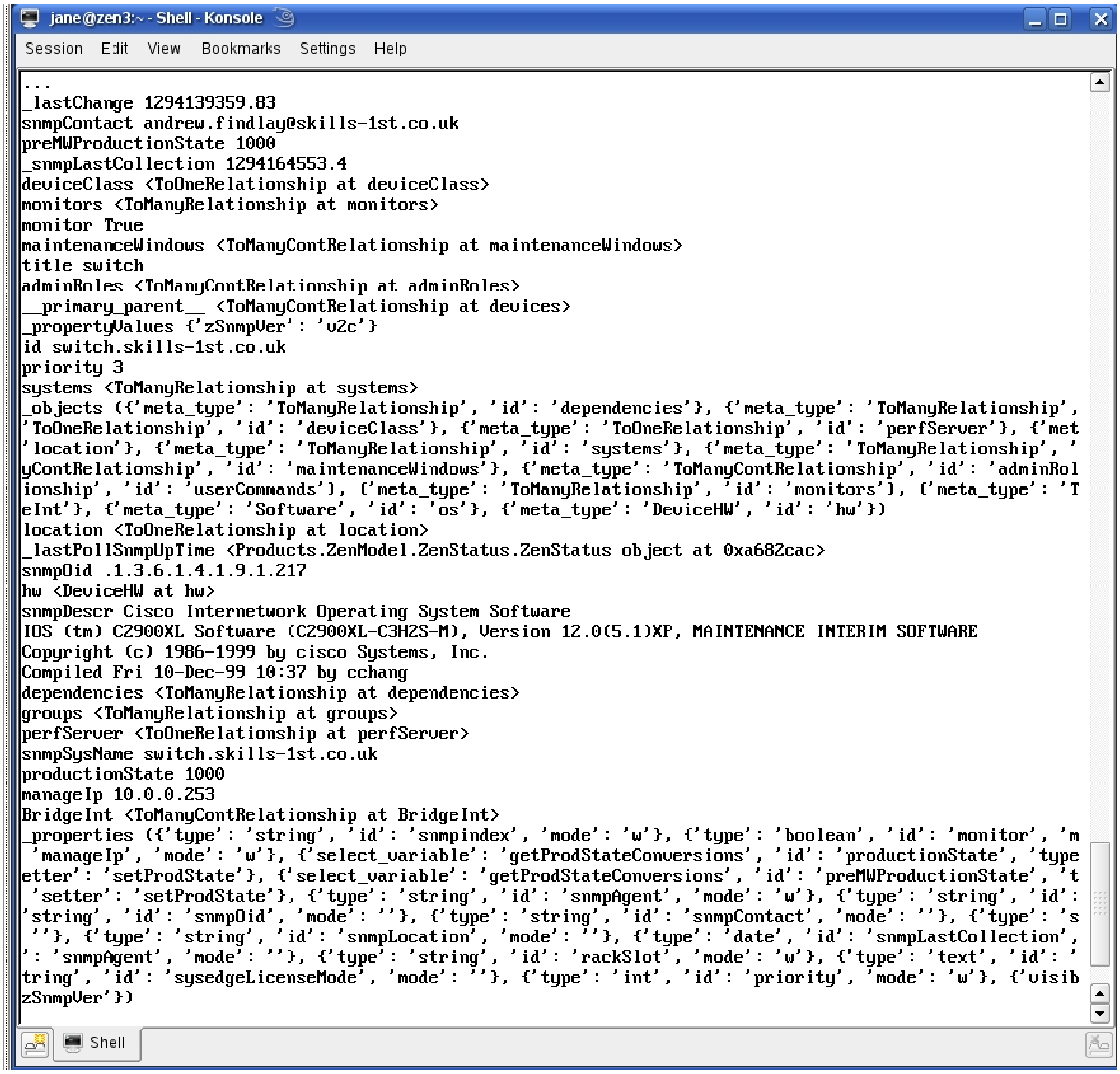
```
    def getRemoteDevice(self):
        """
        return the remote device object for this bridge port. If any are
        returned based on the MAC query we take the first one assuming that
        MACs are unique to devices (even though they aren't on interfaces)
        """
        intobj = self._getInterfaces()
        if len(intobj) > 0 and intobj[0].device():
            return intobj[0].device().urlLink()
```

### 4.3.6 Testing with the zendmd utility

One way to find what attributes of a device are available, is to use the Zenoss *zendmd* utility and run a small series of Python commands:

```
zendmd
>>> dev=find('switch.skills-1st.co.uk')
>>> for key,value in dev.__dict__.items():
...   print key,value
...
```

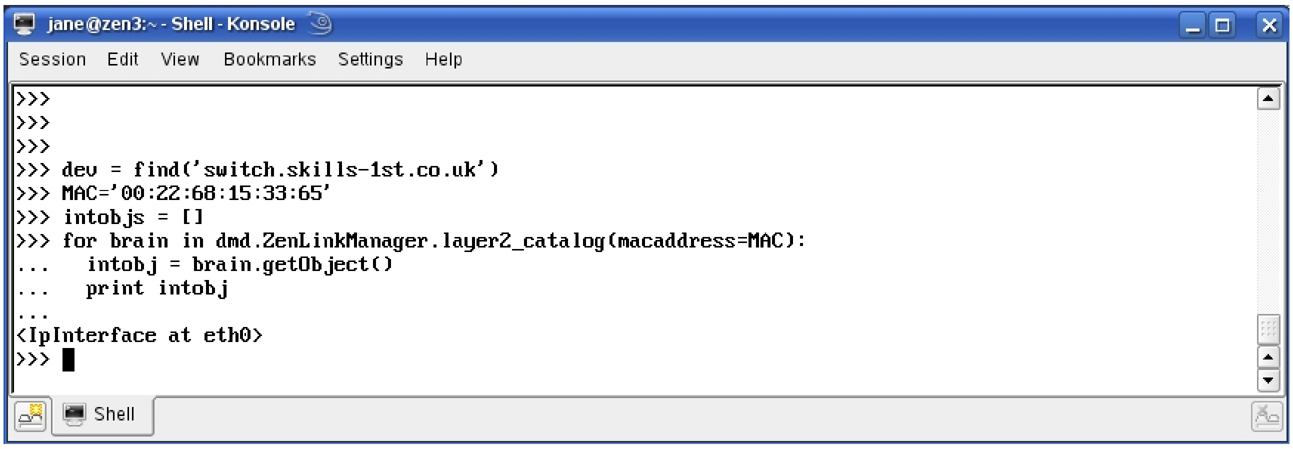
Note that >>> is the zendmd prompt and . . . indicates that a new level of indentation is required. A blank line ends the code and runs the Python, delivering results similar to those shown in Figure 38.



```
...
_lastChange 1294139359.83
snmpContact andrew.findlay@skills-1st.co.uk
preMWProductionState 1000
_snmpLastCollection 1294164553.4
deviceClass <ToOneRelationship at deviceClass>
monitors <ToManyRelationship at monitors>
monitor True
maintenanceWindows <ToManyContRelationship at maintenanceWindows>
title switch
adminRoles <ToManyContRelationship at adminRoles>
__primary_parent__ <ToManyContRelationship at devices>
_propertyValues {'zSnmpVer': 'v2c'}
id switch.skills-1st.co.uk
priority 3
systems <ToManyRelationship at systems>
_objects ({'meta_type': 'ToManyRelationship', 'id': 'dependencies'}, {'meta_type': 'ToManyRelationship',
'ToOneRelationship', 'id': 'deviceClass'}, {'meta_type': 'ToOneRelationship', 'id': 'perfServer'}, {'met
'location'}, {'meta_type': 'ToManyRelationship', 'id': 'systems'}, {'meta_type': 'ToManyRelationship', '
yContRelationship', 'id': 'maintenanceWindows'}, {'meta_type': 'ToManyContRelationship', 'id': 'adminRol
ionship', 'id': 'userCommands'}, {'meta_type': 'ToManyRelationship', 'id': 'monitors'}, {'meta_type': 'T
eInt'}, {'meta_type': 'Software', 'id': 'os'}, {'meta_type': 'DeviceHW', 'id': 'hw'})
location <ToOneRelationship at location>
_lastPollSnmpUpTime <Products.ZenModel.ZenStatus.ZenStatus object at 0xa682cac>
snmpOid .1.3.6.1.4.1.9.1.217
hw <DeviceHW at hw>
snmpDescr Cisco Internetwork Operating System Software
IOS (tm) C2900XL Software (C2900XL-C3H2S-M), Version 12.0(5.1)XP, MAINTENANCE INTERIM SOFTWARE
Copyright (c) 1986-1999 by cisco Systems, Inc.
Compiled Fri 10-Dec-99 10:37 by cchang
dependencies <ToManyRelationship at dependencies>
groups <ToManyRelationship at groups>
perfServer <ToOneRelationship at perfServer>
snmpSysName switch.skills-1st.co.uk
productionState 1000
manageIp 10.0.0.253
BridgeInt <ToManyContRelationship at BridgeInt>
_properties ({'type': 'string', 'id': 'snmpindex', 'mode': 'w'}, {'type': 'boolean', 'id': 'monitor', 'm
 'manageIp', 'mode': 'w'}, {'select_variable': 'getProdStateConversions', 'id': 'productionState', 'type
etter': 'setProdState'}, {'select_variable': 'getProdStateConversions', 'id': 'preMWProductionState', 't
 'setter': 'setProdState'}, {'type': 'string', 'id': 'snmpAgent', 'mode': 'w'}, {'type': 'string', 'id':
'string', 'id': 'snmpOid', 'mode': ''}, {'type': 'string', 'id': 'snmpContact', 'mode': ''}, {'type': 's
 ''}, {'type': 'string', 'id': 'snmpLocation', 'mode': ''}, {'type': 'date', 'id': 'snmpLastCollection',
': 'snmpAgent', 'mode': ''}, {'type': 'string', 'id': 'rackSlot', 'mode': 'w'}, {'type': 'text', 'id': '
tring', 'id': 'sysedgeLicenseMode', 'mode': ''}, {'type': 'int', 'id': 'priority', 'mode': 'w'}, {'visib
zSnmpVer'})
```

*Figure 38: Output of zendmd commands to print attributes of the device switch.skills-1st.co.uk*

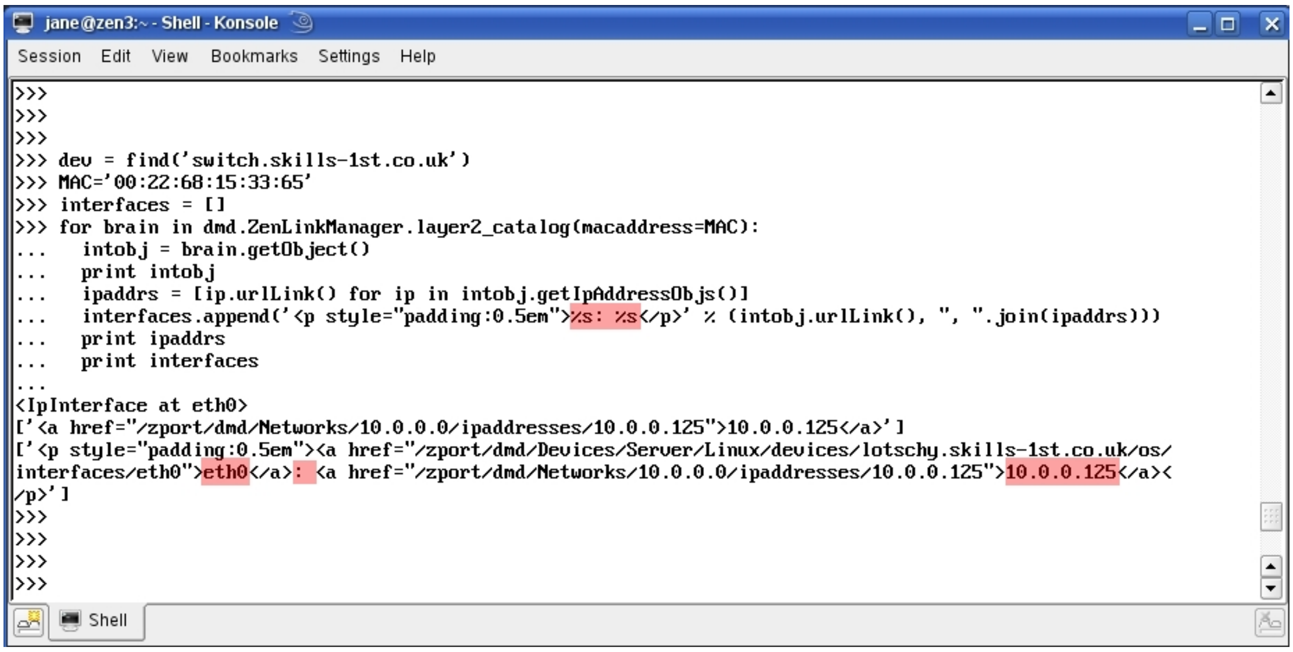
zendmd can be used to test snippets of code. Note that command recall is generally available in zendmd on the up arrow key.


jane@zen3:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
>>>
>>>
>>>
>>> dev = find('switch.skills-1st.co.uk')
>>> MAC='00:22:68:15:33:65'
>>> intobjs = []
>>> for brain in dmd.ZenLinkManager.layer2_catalog(macaddress=MAC):
...   intobj = brain.getObject()
...   print intobj
...
<IpInterface at eth0>
>>> 
```

*Figure 39: Testing snippets of code with zendmd - getting an interface object*

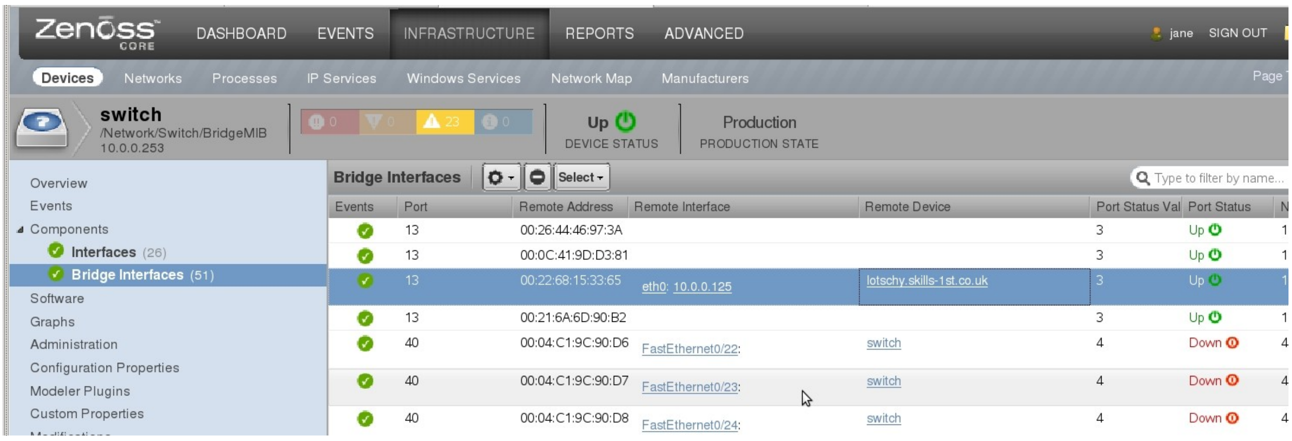
You often need to simplify your zendmd tests to small units; in Figure 39 and Figure 40 the MAC address of a known device has been hardcoded into the variable *MAC*.


jane@zen3:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
>>>
>>>
>>>
>>> dev = find('switch.skills-1st.co.uk')
>>> MAC='00:22:68:15:33:65'
>>> interfaces = []
>>> for brain in dmd.ZenLinkManager.layer2_catalog(macaddress=MAC):
...   intobj = brain.getObject()
...   print intobj
...   ipaddrs = [ip.urlLink() for ip in intobj.getIpAddressObjs()]
...   interfaces.append('<p style="padding:0.5em">%s: %s</p>' % (intobj.urlLink(), ", ".join(ipaddrs)))
...   print ipaddrs
...   print interfaces
...
<IpInterface at eth0>
['<a href="/zport/dmd/Networks/10.0.0.0/ipaddresses/10.0.0.125">10.0.0.125</a>']
['<p style="padding:0.5em"><a href="/zport/dmd/Devices/Server/Linux/devices/lotschy.skills-1st.co.uk/os/
interfaces/eth0">eth0</a>: <a href="/zport/dmd/Networks/10.0.0.0/ipaddresses/10.0.0.125">10.0.0.125</a><
/p>']
>>>
>>>
>>>
>>>
```

*Figure 40: Testing snippets of code with zendmd – understanding the Remote Interface url links*

Compare the print output in Figure 40 with the screenshot below that demonstrates the Bridge Interfaces GUI for the device with MAC address 00:22:68:15:33:65. The *Remote Interface* column shows *eth0*, followed by colon and a space, followed by the IP interface address of *10.0.0.125* and both these elements are url links to other parts of Zenoss.



| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | N |
|---|---|---|---|---|---|---|---|
| ✓ | 13 | 00:26:44:46:97:3A | | | 3 | Up | 1 |
| ✓ | 13 | 00:0C:41:9D:D3:81 | | | 3 | Up | 1 |
| ✓ | 13 | 00:22:68:15:33:65 | eth0: 10.0.0.125 | lotschy.skills-1st.co.uk | 3 | Up | 1 |
| ✓ | 13 | 00:21:6A:6D:90:B2 | | | 3 | Up | 1 |
| ✓ | 40 | 00:04:C1:9C:90:D6 | FastEthernet0/22: | switch | 4 | Down | 4 |
| ✓ | 40 | 00:04:C1:9C:90:D7 | FastEthernet0/23: | switch | 4 | Down | 4 |
| ✓ | 40 | 00:04:C1:9C:90:D8 | FastEthernet0/24: | switch | 4 | Down | 4 |

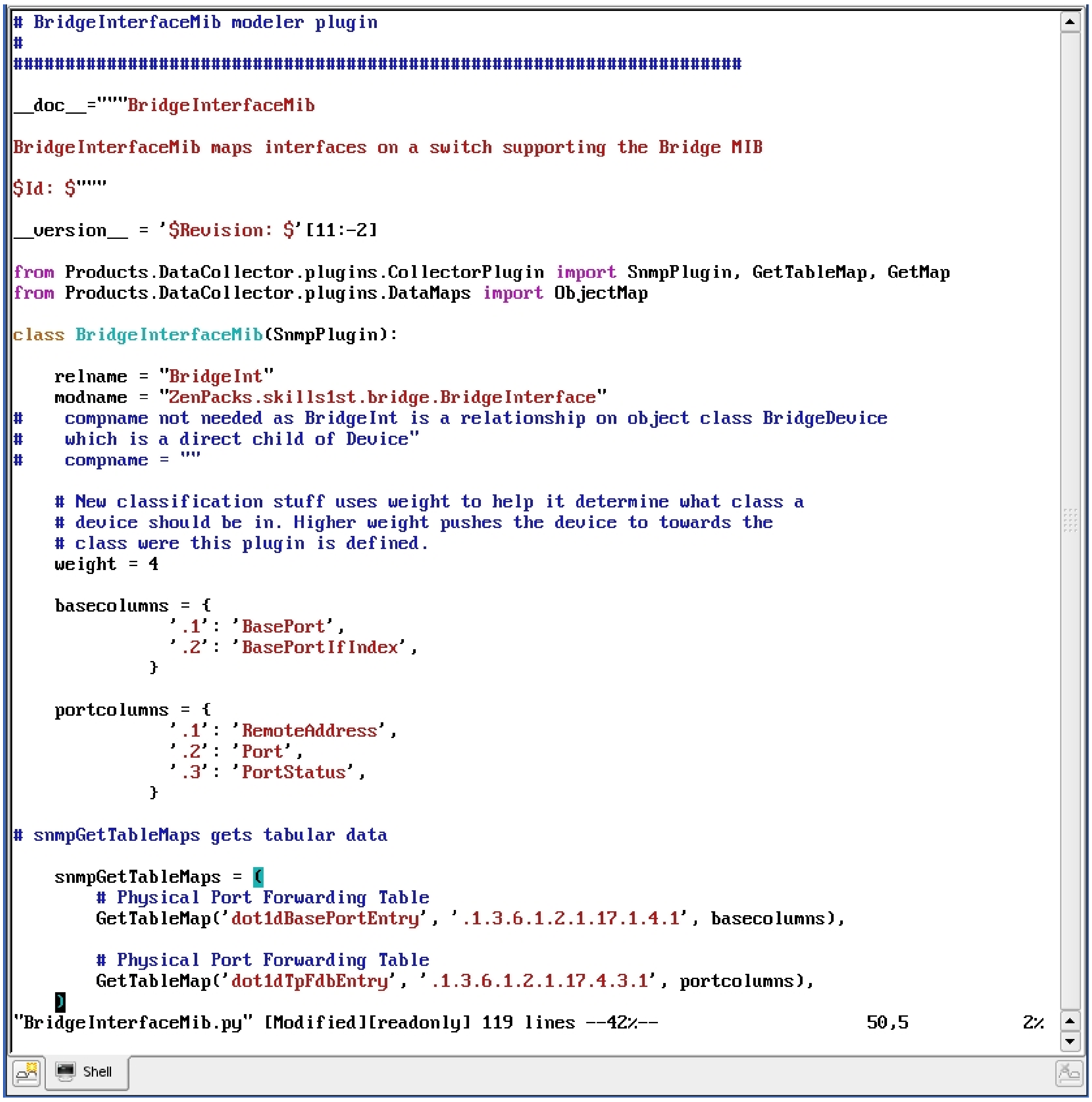
*Figure 41: Bridge Interfaces GUI highlighting Remote Interface link for given MAC address*

### 4.3.7 Creating the modeler plugin files

This ZenPack has two modeler plugin files, residing under the base ZenPack directory under the modeler/plugins subdirectory hierarchy. They are:

| | |
|---|---|
| • BridgeInterfaceMib.py | gets port data for each switch port |
| • BridgeDeviceMib.py | gets scalar data for the switch device |

These names can be anything but should obviously be relevant. The only place where these names appear is when a device or device class has its Collector Plugins configured from the Zenoss GUI. The purpose of a modeler plugin is to **map** collected data into the attributes of Zenoss objects.



```
# BridgeInterfaceMib modeler plugin
#
######################################################################

__doc__="""BridgeInterfaceMib

BridgeInterfaceMib maps interfaces on a switch supporting the Bridge MIB

$Id: $"""

__version__ = '$Revision: $'[11:-2]

from Products.DataCollector.plugins.CollectorPlugin import SnmpPlugin, GetTableMap, GetMap
from Products.DataCollector.plugins.DataMaps import ObjectMap

class BridgeInterfaceMib(SnmpPlugin):

    relname = "BridgeInt"
    modname = "ZenPacks.skills1st.bridge.BridgeInterface"
#    compname not needed as BridgeInt is a relationship on object class BridgeDevice
#    which is a direct child of Device"
#    compname = ""

    # New classification stuff uses weight to help it determine what class a
    # device should be in. Higher weight pushes the device to towards the
    # class were this plugin is defined.
    weight = 4

    basecolumns = {
                 '.1': 'BasePort',
                 '.2': 'BasePortIfIndex',
             }

    portcolumns = {
                 '.1': 'RemoteAddress',
                 '.2': 'Port',
                 '.3': 'PortStatus',
             }

# snmpGetTableMaps gets tabular data

    snmpGetTableMaps = (
        # Physical Port Forwarding Table
        GetTableMap('dot1dBasePortEntry', '.1.3.6.1.2.1.17.1.4.1', basecolumns),

        # Physical Port Forwarding Table
        GetTableMap('dot1dTpFdbEntry', '.1.3.6.1.2.1.17.4.3.1', portcolumns),
    )
"BridgeInterfaceMib.py" [Modified][readonly] 119 lines --42%--               50,5          2%
```

*Figure 42: BridgeInterfaceMib modeler plugin (part 1) with SNMP data to be collected*

The first part of the BridgeInterfaceMib modeler plugin code imports some standard Zenoss utilities for getting SNMP information and formatting it.

```
from Products.DataCollector.plugins.CollectorPlugin import SnmpPlugin, GetTableMap, GetMap
from Products.DataCollector.plugins.DataMaps import ObjectMap
```

Note that the modeler plugin, BridgeInterfaceMib, is itself defined as an object class which derives from the standard SnmpPlugin modeler. The modeler must be activated for a device or device class (from the *Modeler Plugins* left-hand menu) – it cannot be directly activated for a device component such as a port on a switch. Hence, the *relname* and *modname* directives specify that the data is to be applied to a relationship of the device, the component object class being specified by the modname line.

```
    class BridgeInterfaceMib(SnmpPlugin):

        relname = "BridgeInt"
        modname = "ZenPacks.skills1st.bridge.BridgeInterface"
```

In other words, the modeler is applied to a device of object class BridgeDevice but the data will be mapped to the contained relationship called BridgeInt whose data attributes are specified by ZenPacks.skills1st.bridge.BridgeInterface; this comes down to populating the unique RemoteAddress, Port, PortIfIndex and PortStatus attributes.

The next part of the modeler plugin specifies SNMP data tables and ObjectIDs (OIDs) to collect.

```
        basecolumns = {
                   '.1': 'BasePort',
                   '.2': 'BasePortIfIndex',
                 }

        portcolumns = {
                   '.1': 'RemoteAddress',
                   '.2': 'Port',
                   '.3': 'PortStatus',
                 }

    # snmpGetTableMaps gets tabular data

        snmpGetTableMaps = (
            # Physical Port Forwarding Table
            GetTableMap('dot1dBasePortEntry', '.1.3.6.1.2.1.17.1.4.1',
    basecolumns),

            # Physical Port Forwarding Table
            GetTableMap('dot1dTpFdbEntry', '.1.3.6.1.2.1.17.4.3.1',
    portcolumns),
        )
```

The GetTableMap standard Zenoss function takes three parameters:

- A table name you'll be using later (this can be anything but it is helpful if it matches the name of the SNMP table)
- The OID of the SNMP table
- A dictionary of "OID-endings" and column names (OID-endings being the keys, used later)

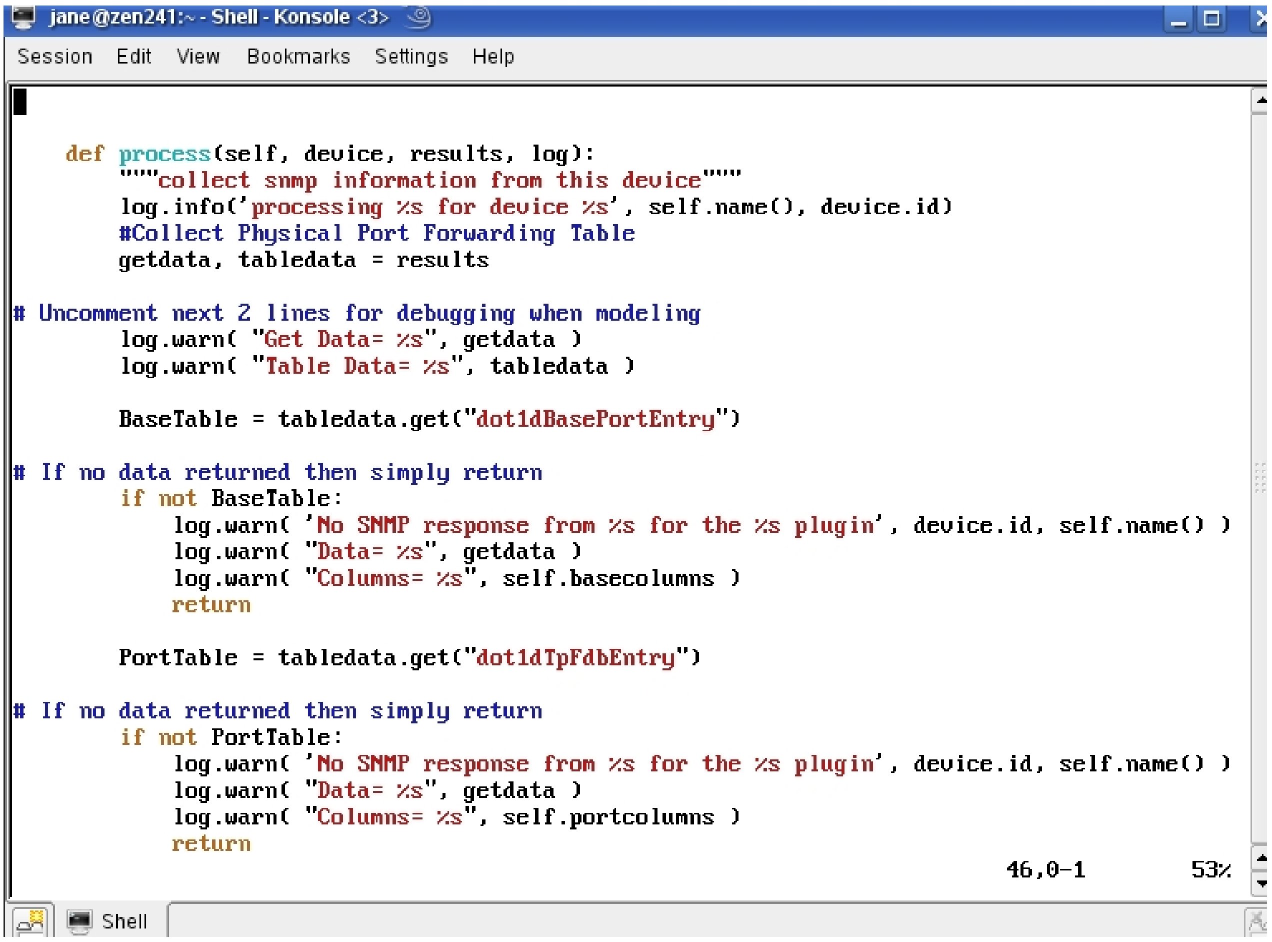
If there are only one or two OIDs required, it is perfectly possible to code them directly as part of GetTableMap. It is also possible to specify the OID-ending as more than the last digit. For example, the following code has the same effect as the first GetTableMap stanza above.

```
    GetTableMap('dot1dBasePortEntry', '.1.3.6.1.2.1.17.1.4',
        {'1.1': 'BasePort',
         '1.2': 'BasePortIfIndex',
        }
    ),
```

It is usually clearer and more convenient to specify the dictionary of "OID-endings" and column names separately as shown above with basecolumns.

The snmpGetTableMaps function can get one or more SNMP tables of data.

The only mandatory function required in a modeler plugin is the *process()* function.


jane@zen241:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
Shell

```
    def process(self, device, results, log):
        """collect snmp information from this device"""
        log.info('processing %s for device %s', self.name(), device.id)
        #Collect Physical Port Forwarding Table
        getdata, tabledata = results

# Uncomment next 2 lines for debugging when modeling
        log.warn( "Get Data= %s", getdata )
        log.warn( "Table Data= %s", tabledata )

        BaseTable = tabledata.get("dot1dBasePortEntry")

# If no data returned then simply return
        if not BaseTable:
            log.warn( 'No SNMP response from %s for the %s plugin', device.id, self.name() )
            log.warn( "Data= %s", getdata )
            log.warn( "Columns= %s", self.basecolumns )
            return

        PortTable = tabledata.get("dot1dTpFdbEntry")

# If no data returned then simply return
        if not PortTable:
            log.warn( 'No SNMP response from %s for the %s plugin', device.id, self.name() )
            log.warn( "Data= %s", getdata )
            log.warn( "Columns= %s", self.portcolumns )
            return
                                                                      46,0-1        53%
```

*Figure 43: BridgeInterfaceMib modeler plugin (part 2) showing data collection and error checking*

The part that actually gets the data is the line:

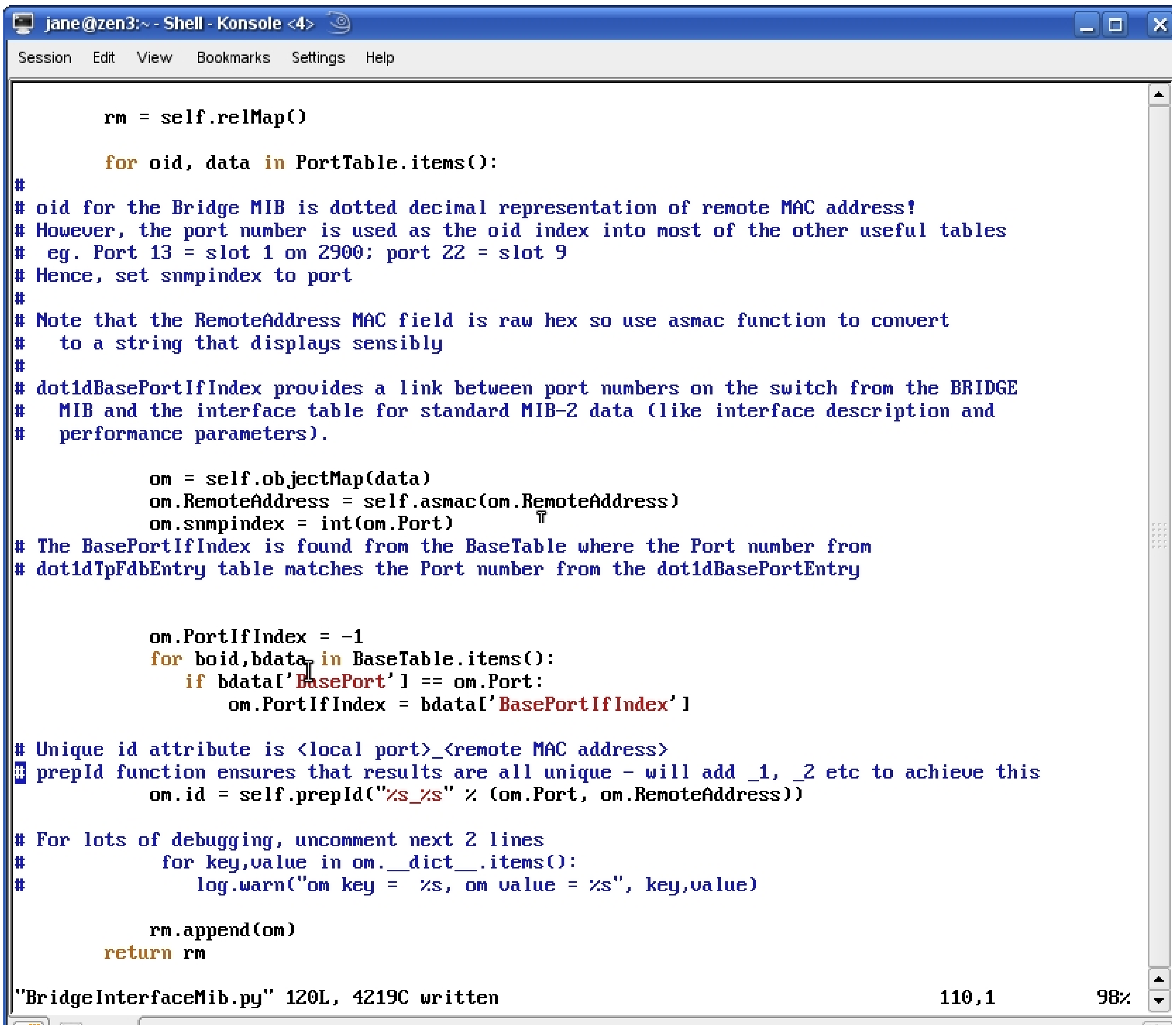
```
    getdata, tabledata = results
```

Scalar data is populated into *getdata*; table data is populated into *tabledata*. Debugging can be provided using *log* statements with different severities such as log.*info* and log.*warn*.

```
    log.info('processing %s for device %s', self.name(), device.id)
```

Remember that snmpGetTableMaps retrieves two tables of data into the variables dot1dBasePortEntry and dot1dTpFdbEntry. The second half of Figure 43 checks that SNMP data was actually retrieved (as the device may, for example, have been down on a modeler cycle). If either table is not populated then logging is produced and the process function simply returns.

The last part of the modeler plugin code creates a relationship mapping that will contain entries for each object that represents a port on the device.

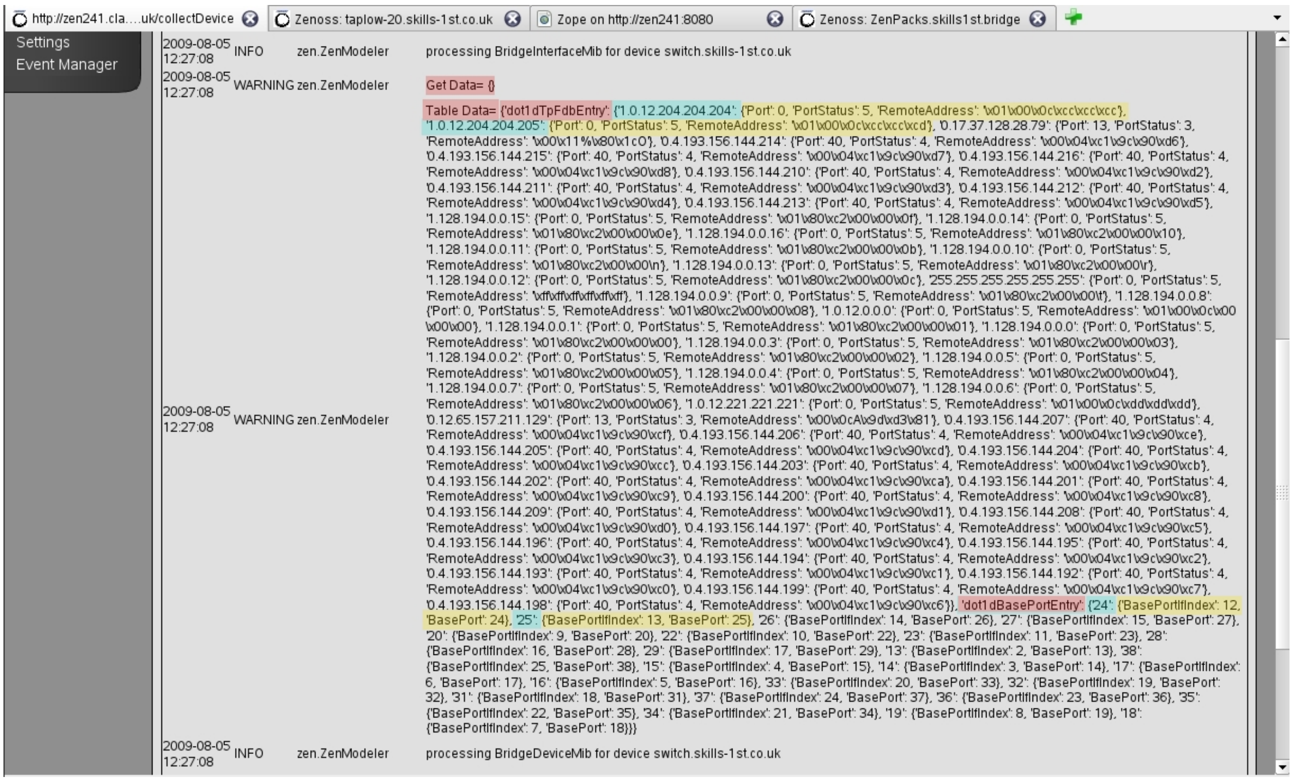

jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
        rm = self.relMap()

        for oid, data in PortTable.items():
#
# oid for the Bridge MIB is dotted decimal representation of remote MAC address!
# However, the port number is used as the oid index into most of the other useful tables
#  eg. Port 13 = slot 1 on 2900; port 22 = slot 9
# Hence, set snmpindex to port
#
# Note that the RemoteAddress MAC field is raw hex so use asmac function to convert
#   to a string that displays sensibly
#
# dot1dBasePortIfIndex provides a link between port numbers on the switch from the BRIDGE
#   MIB and the interface table for standard MIB-2 data (like interface description and
#   performance parameters).

            om = self.objectMap(data)
            om.RemoteAddress = self.asmac(om.RemoteAddress)
            om.snmpindex = int(om.Port)
# The BasePortIfIndex is found from the BaseTable where the Port number from
# dot1dTpFdbEntry table matches the Port number from the dot1dBasePortEntry


            om.PortIfIndex = -1
            for boid,bdata in BaseTable.items():
               if bdata['BasePort'] == om.Port:
                   om.PortIfIndex = bdata['BasePortIfIndex']

# Unique id attribute is <local port>_<remote MAC address>
# prepId function ensures that results are all unique - will add _1, _2 etc to achieve this
            om.id = self.prepId("%s_%s" % (om.Port, om.RemoteAddress))

# For lots of debugging, uncomment next 2 lines
#             for key,value in om.__dict__.items():
#                log.warn("om key =  %s, om value = %s", key,value)

            rm.append(om)
        return rm

"BridgeInterfaceMib.py" 120L, 4219C written                        110,1        98%
```

Figure 44: *BridgeInterfaceMib modeler plugin (part 3) mapping and modifying SNMP data onto objects*

Remember that the GetTableMap delivers a table (strictly a Python dictionary). The two fields of the dictionary are the OID and the data; the data itself is also a dictionary containing column names and values. To see what is actually delivered, make sure that the following lines are uncommented and then model a switch device from the *Manage -> Model* device menu.

```
# Uncomment next 2 lines for debugging when modeling
        log.warn( "Get Data= %s", getdata )
        log.warn( "Table Data= %s", tabledata )
```



http://zen241.cla....uk/collectDevice
Zenoss: taplow-20.skills-1st.co.uk
Zope on http://zen241:8080
Zenoss: ZenPacks.skills1st.bridge
Settings
Event Manager

```
2009-08-05 12:27:08  INFO     zen.ZenModeler  processing BridgeInterfaceMib for device switch.skills-1st.co.uk
2009-08-05 12:27:08  WARNING  zen.ZenModeler  Get Data= {}
2009-08-05 12:27:08  WARNING  zen.ZenModeler  Table Data= {'dot1dTpFdbEntry': {'1.0.12.204.204.204': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x00\x0c\xcc\xcc\xcc'},
'1.0.12.204.204.205': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x00\x0c\xcc\xcc\xcd'}, '0.17.37.128.28.79': {'Port': 13, 'PortStatus': 3,
'RemoteAddress': '\x00\x11%\x80\x1cO'}, '0.4.193.156.144.214': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd6'},
'0.4.193.156.144.215': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd7'}, '0.4.193.156.144.216': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd8'}, '0.4.193.156.144.210': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd2'},
'0.4.193.156.144.211': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd3'}, '0.4.193.156.144.212': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd4'}, '0.4.193.156.144.213': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd5'},
'1.128.194.0.0.15': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x0f'}, '1.128.194.0.0.14': {'Port': 0, 'PortStatus': 5,
'RemoteAddress': '\x01\x80\xc2\x00\x00\x0e'}, '1.128.194.0.0.16': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x10'},
'1.128.194.0.0.11': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x0b'}, '1.128.194.0.0.10': {'Port': 0, 'PortStatus': 5,
'RemoteAddress': '\x01\x80\xc2\x00\x00\n'}, '1.128.194.0.0.13': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\r'},
'1.128.194.0.0.12': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x0c'}, '255.255.255.255.255.255': {'Port': 0, 'PortStatus': 5,
'RemoteAddress': '\xff\xff\xff\xff\xff\xff'}, '1.128.194.0.0.9': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\t'}, '1.128.194.0.0.8':
{'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x08'}, '1.0.12.0.0.0': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x00\x0c\x00
\x00\x00'}, '1.128.194.0.0.1': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x01'}, '1.128.194.0.0.0': {'Port': 0, 'PortStatus': 5,
'RemoteAddress': '\x01\x80\xc2\x00\x00\x00'}, '1.128.194.0.0.3': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x03'},
'1.128.194.0.0.2': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x02'}, '1.128.194.0.0.5': {'Port': 0, 'PortStatus': 5,
'RemoteAddress': '\x01\x80\xc2\x00\x00\x05'}, '1.128.194.0.0.4': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x04'},
'1.128.194.0.0.7': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x80\xc2\x00\x00\x07'}, '1.128.194.0.0.6': {'Port': 0, 'PortStatus': 5,
'RemoteAddress': '\x01\x80\xc2\x00\x00\x06'}, '1.0.12.221.221.221': {'Port': 0, 'PortStatus': 5, 'RemoteAddress': '\x01\x00\x0c\xdd\xdd\xdd'},
'0.12.65.157.211.129': {'Port': 13, 'PortStatus': 3, 'RemoteAddress': '\x00\x0cA\x9d\xd3\x81'}, '0.4.193.156.144.207': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xcf'}, '0.4.193.156.144.206': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xce'},
'0.4.193.156.144.205': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xcd'}, '0.4.193.156.144.204': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xcc'}, '0.4.193.156.144.203': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xcb'},
'0.4.193.156.144.202': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xca'}, '0.4.193.156.144.201': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc9'}, '0.4.193.156.144.200': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc8'},
'0.4.193.156.144.209': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd1'}, '0.4.193.156.144.208': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xd0'}, '0.4.193.156.144.197': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc5'},
'0.4.193.156.144.196': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc4'}, '0.4.193.156.144.195': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc3'}, '0.4.193.156.144.194': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc2'},
'0.4.193.156.144.193': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc1'}, '0.4.193.156.144.192': {'Port': 40, 'PortStatus': 4,
'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc0'}, '0.4.193.156.144.199': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc7'},
'0.4.193.156.144.198': {'Port': 40, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x9c\x90\xc6'}}, 'dot1dBasePortEntry': {'24': {'BasePortIfIndex': 12,
'BasePort': 24}, '25': {'BasePortIfIndex': 13, 'BasePort': 25}, '26': {'BasePortIfIndex': 14, 'BasePort': 26}, '27': {'BasePortIfIndex': 15, 'BasePort': 27},
'20': {'BasePortIfIndex': 9, 'BasePort': 20}, '22': {'BasePortIfIndex': 10, 'BasePort': 22}, '23': {'BasePortIfIndex': 11, 'BasePort': 23}, '28':
{'BasePortIfIndex': 16, 'BasePort': 28}, '29': {'BasePortIfIndex': 17, 'BasePort': 29}, '13': {'BasePortIfIndex': 2, 'BasePort': 13}, '38':
{'BasePortIfIndex': 25, 'BasePort': 38}, '15': {'BasePortIfIndex': 4, 'BasePort': 15}, '14': {'BasePortIfIndex': 3, 'BasePort': 14}, '17': {'BasePortIfIndex':
6, 'BasePort': 17}, '16': {'BasePortIfIndex': 5, 'BasePort': 16}, '33': {'BasePortIfIndex': 20, 'BasePort': 33}, '32': {'BasePortIfIndex': 19, 'BasePort':
32}, '31': {'BasePortIfIndex': 18, 'BasePort': 31}, '37': {'BasePortIfIndex': 24, 'BasePort': 37}, '36': {'BasePortIfIndex': 23, 'BasePort': 36}, '35':
{'BasePortIfIndex': 22, 'BasePort': 35}, '34': {'BasePortIfIndex': 21, 'BasePort': 34}, '19': {'BasePortIfIndex': 8, 'BasePort': 19}, '18':
{'BasePortIfIndex': 7, 'BasePort': 18}}}
2009-08-05 12:27:08  INFO     zen.ZenModeler  processing BridgeDeviceMib for device switch.skills-1st.co.uk
```

*Figure 45: Debug output for BridgeInterfaceMib modeler plugin*

The BridgeInterfaceMib modeler plugin doesn't, in fact, get scalar data, so the the getdata dictionary is empty (ie. {} ). snmpGetTableMaps delivers two tables (Python dictionaries) – dot1dTpFdbEntry and dot1dBasePortEntry; these are all shown highlighted in red in Figure 45. Each of dot1dTpFdbEntry and dot1dBasePortEntry comprises a dictionary with OID and data components. The first few OID values are highlighted in blue for each table. The data component is itself a dictionary with column names and values; these are highlighted in yellow.

So, the lines:

```
for oid, data in PortTable.items():
  om=self.objectMap(data)
```

cycles through each of the OID, data sets of values in the PortTable, mapping the data values to the attributes of the BridgeInterface object; Port, PortStatus and RemoteAddress.

Note in Figure 45 that the MAC address is in hex format. To display this for users, it needs converting to a string-type representation so the delivered value of the RemoteAddress is converted using the Python *asmac* function, replacing the RemoteAddress value on the object.

```
om.RemoteAddress = self.asmac(om.RemoteAddress)
```

The ZenPack only defined four unique attributes for the BridgeInterface object in the object class file BridgeInterface.py:

- RemoteAddress
- Port

- PortIfIndex
- PortStatus

However, it also inherited attributes as a DeviceComponent and ManagedEntity and thus has other attributes, including:

- id
- snmpindex

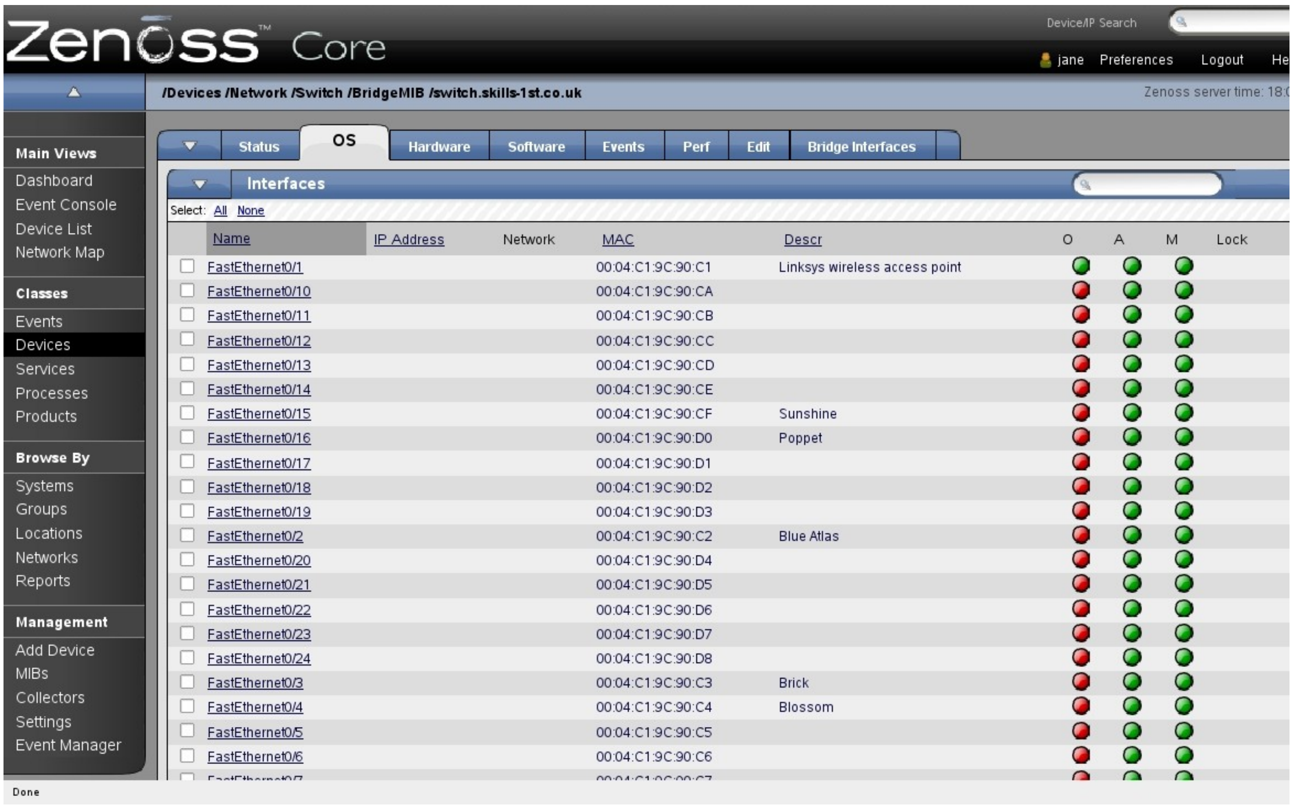
The *id* should be a unique and meaningful identifier – it is set to <local port number>_<remote MAC address> with the standard *prepId* function used to ensure uniqueness of names.

*snmpindex* is used when performance data is configured using Zenoss templates and provides the instance to collect for any given SNMP OID. Most of the useful SNMP data to do with switch ports is actually indexed using the value of the port number (remember for the test Catalyst 2900 switch, the values of Port representing real interfaces run from 13 to 38 – you can see the values for a real active port in Figure 45 right in the middle, opposite WARNING zen.ZenModeler). Hence, the snmpindex attribute is set to the Port value, having first converted the raw data to an integer type.

```
    om.snmpindex = int(om.Port)
```

Note that many modeler plugins use the OID value from the tabledata as the snmpindex but this is only useful if that OID does actually represent a useful SNMP index. The OID value that we have delivered (highlighted in blue in Figure 45) is the decimal representation of a MAC address and is not useful as an instance for collecting performance information. More on this topic later.

A switch discovered by Zenoss will automatically gather information on each of the ports, using information from the MIB-2 MIB. This doesn't provide much port-level information but it does provide some. The interfaces are indexed using the interfaces table of MIB-2.

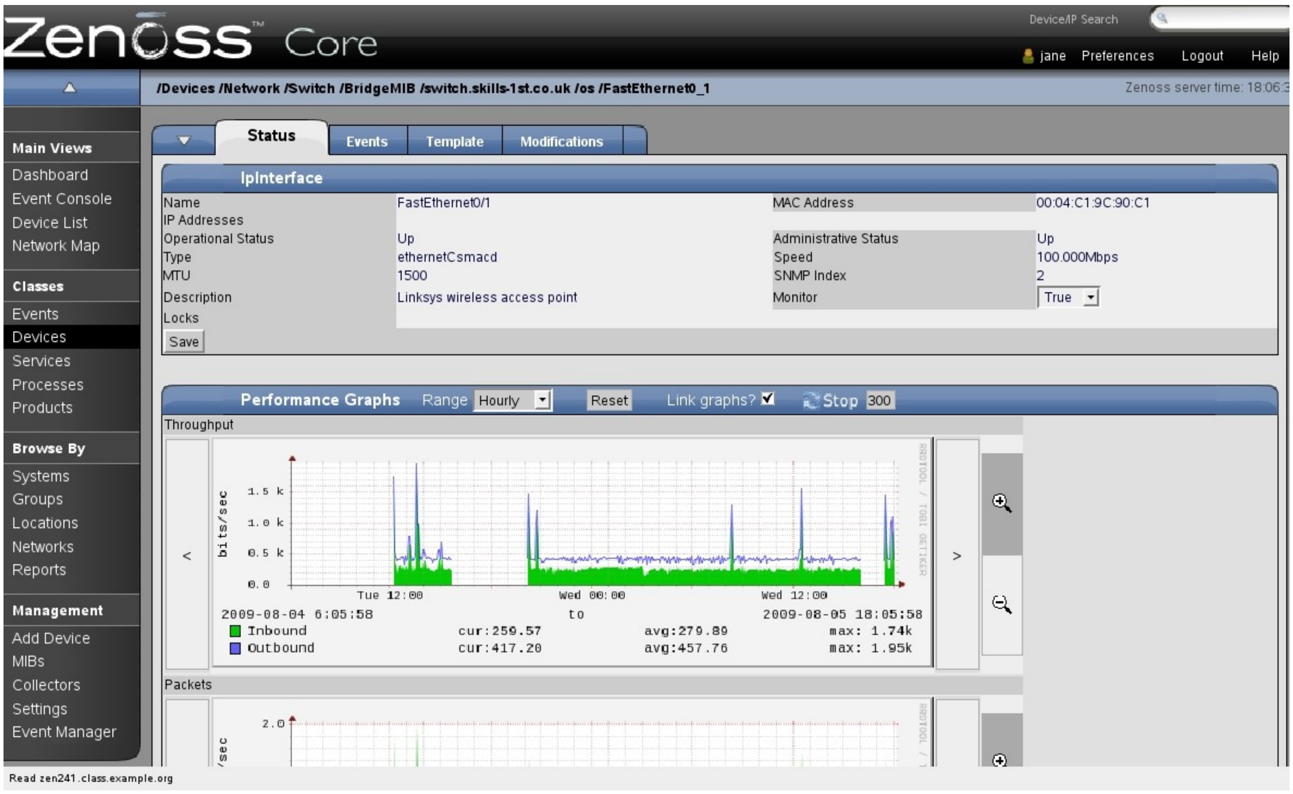


/Devices /Network /Switch /BridgeMIB /switch.skills-1st.co.uk

| Name | IP Address | Network | MAC | Descr | O | A | M | Lock |
|---|---|---|---|---|---|---|---|---|
| FastEthernet0/1 | | | 00:04:C1:9C:90:C1 | Linksys wireless access point | 🟢 | 🟢 | 🟢 | |
| FastEthernet0/10 | | | 00:04:C1:9C:90:CA | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/11 | | | 00:04:C1:9C:90:CB | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/12 | | | 00:04:C1:9C:90:CC | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/13 | | | 00:04:C1:9C:90:CD | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/14 | | | 00:04:C1:9C:90:CE | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/15 | | | 00:04:C1:9C:90:CF | Sunshine | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/16 | | | 00:04:C1:9C:90:D0 | Poppet | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/17 | | | 00:04:C1:9C:90:D1 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/18 | | | 00:04:C1:9C:90:D2 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/19 | | | 00:04:C1:9C:90:D3 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/2 | | | 00:04:C1:9C:90:C2 | Blue Atlas | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/20 | | | 00:04:C1:9C:90:D4 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/21 | | | 00:04:C1:9C:90:D5 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/22 | | | 00:04:C1:9C:90:D6 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/23 | | | 00:04:C1:9C:90:D7 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/24 | | | 00:04:C1:9C:90:D8 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/3 | | | 00:04:C1:9C:90:C3 | Brick | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/4 | | | 00:04:C1:9C:90:C4 | Blossom | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/5 | | | 00:04:C1:9C:90:C5 | | 🔴 | 🟢 | 🟢 | |
| FastEthernet0/6 | | | 00:04:C1:9C:90:C6 | | 🔴 | 🟢 | 🟢 | |

*Figure 46: Standard OS tab for switch device with MIB-2 interfaces in Zenoss 2*

Drilling in to an interface results in both tabular information and performance graphs for bound templates.



/Devices /Network /Switch /BridgeMIB /switch.skills-1st.co.uk /os /FastEthernet0_1

| | | | |
|---|---|---|---|
| Name | FastEthernet0/1 | MAC Address | 00:04:C1:9C:90:C1 |
| IP Addresses | | | |
| Operational Status | Up | Administrative Status | Up |
| Type | ethernetCsmacd | Speed | 100.000Mbps |
| MTU | 1500 | SNMP Index | 2 |
| Description | Linksys wireless access point | Monitor | True |
| Locks | | | |

```
2009-08-04 6:05:58            to            2009-08-05 18:05:58
🟩 Inbound           cur:259.57    avg:279.89      max: 1.74k
🟦 Outbound          cur:417.20    avg:457.76      max: 1.95k
```

*Figure 47: Tabular information and performance graphs for a switch interface from MIB-2 in Zenoss 2*

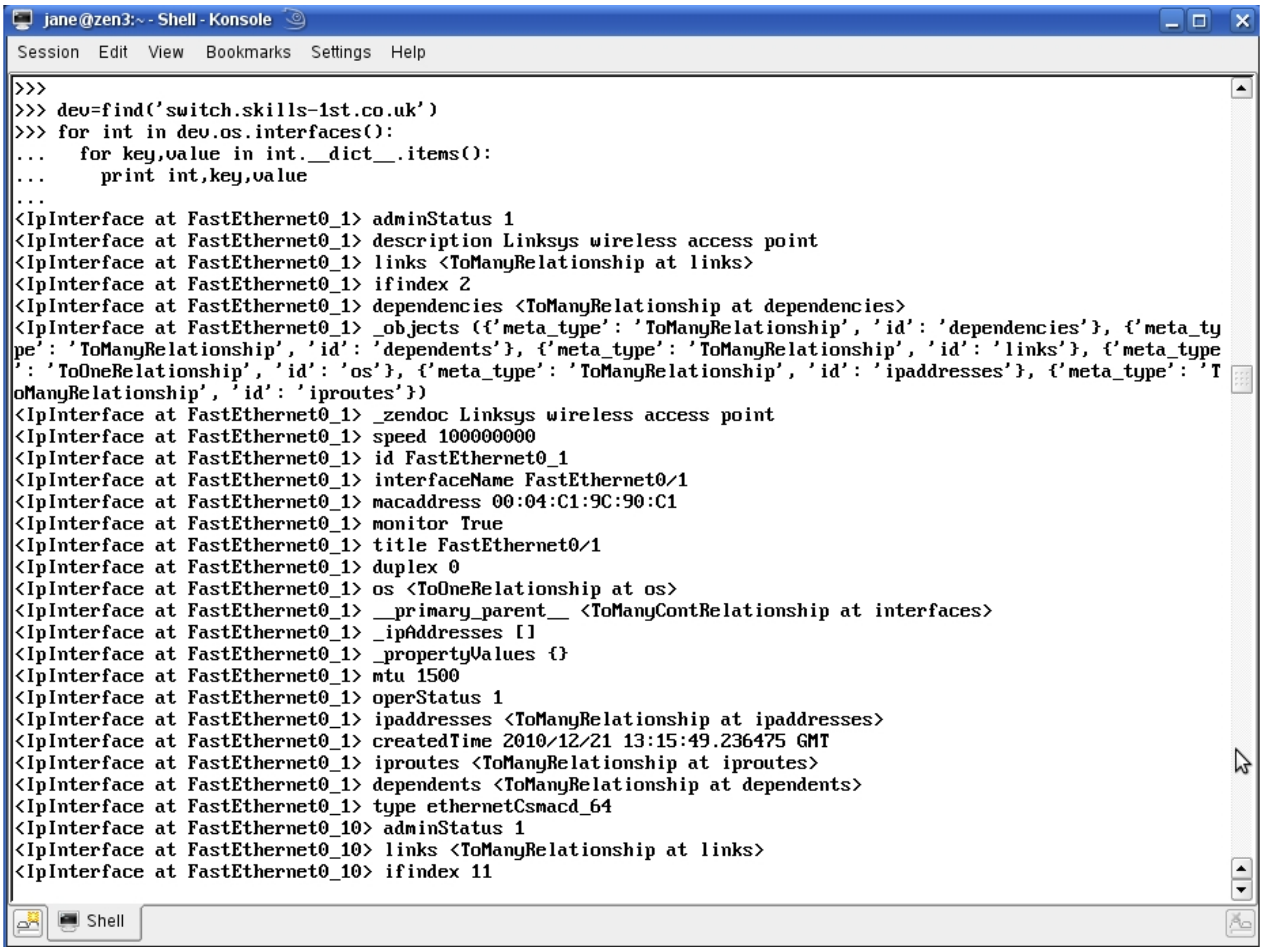
Note in the table at the top of Figure 47 that the SNMP Index for this interface is given as *2*. It is this index number that is delivered by the BRIDGE MIB to coordinate MIB-2 interface information with BRIDGE MIB information. The OID is the BasePortIfIndex from the dot1dBasePortEntry table ( .1.3.6.1.2.1.17.1.4.1.2). . 1.3.6.1.2.1.17.1.4.1.1 (BasePort from the same table) will be the same as the Port OID from the forwarding table (.1.3.6.1.2.1.17.4.3.1.2). The following code delivers the PortIfIndex attribute to the BridgeInterface object, if a valid PortIfIndex exists (it won't for internal and management ports); otherwise PortIfIndex will be -1.

```
# The BasePortIfIndex is found from the BaseTable where  Port number from
# dot1dTpFdbEntry table matches the Port number from the dot1dBasePortEntry

           om.PortIfIndex = -1
            for boid,bdata in BaseTable.items():
               if bdata['BasePort'] == om.Port:
                   om.PortIfIndex = bdata['BasePortIfIndex']
```

Having cycled around these attribute mappings for the data for the first port, the object map is appended to the relationship map, and the next set of port data is processed.

```
            rm.append(om)
        return rm
```

The last two screenshots showing the standard MIB-2 interface information for a switch, are deliberately taken from Zenoss 2. Unfortunately Zenoss 3 delivers almost the same information as Zenoss 2 between the *Graphs* dropdown and the *Details* dropdown but the one piece of information omitted is the SNMP Index. This will be revisited later. One way to see all MIB-2 interface information is to use zendmd.



```
>>>
>>> dev=find('switch.skills-1st.co.uk')
>>> for int in dev.os.interfaces():
...   for key,value in int.__dict__.items():
...     print int,key,value
...
<IpInterface at FastEthernet0_1> adminStatus 1
<IpInterface at FastEthernet0_1> description Linksys wireless access point
<IpInterface at FastEthernet0_1> links <ToManyRelationship at links>
<IpInterface at FastEthernet0_1> ifindex 2
<IpInterface at FastEthernet0_1> dependencies <ToManyRelationship at dependencies>
<IpInterface at FastEthernet0_1> _objects ({'meta_type': 'ToManyRelationship', 'id': 'dependencies'}, {'meta_ty
pe': 'ToManyRelationship', 'id': 'dependents'}, {'meta_type': 'ToManyRelationship', 'id': 'links'}, {'meta_type
': 'ToOneRelationship', 'id': 'os'}, {'meta_type': 'ToManyRelationship', 'id': 'ipaddresses'}, {'meta_type': 'T
oManyRelationship', 'id': 'iproutes'})
<IpInterface at FastEthernet0_1> _zendoc Linksys wireless access point
<IpInterface at FastEthernet0_1> speed 100000000
<IpInterface at FastEthernet0_1> id FastEthernet0_1
<IpInterface at FastEthernet0_1> interfaceName FastEthernet0/1
<IpInterface at FastEthernet0_1> macaddress 00:04:C1:9C:90:C1
<IpInterface at FastEthernet0_1> monitor True
<IpInterface at FastEthernet0_1> title FastEthernet0/1
<IpInterface at FastEthernet0_1> duplex 0
<IpInterface at FastEthernet0_1> os <ToOneRelationship at os>
<IpInterface at FastEthernet0_1> __primary_parent__ <ToManyContRelationship at interfaces>
<IpInterface at FastEthernet0_1> _ipAddresses []
<IpInterface at FastEthernet0_1> _propertyValues {}
<IpInterface at FastEthernet0_1> mtu 1500
<IpInterface at FastEthernet0_1> operStatus 1
<IpInterface at FastEthernet0_1> ipaddresses <ToManyRelationship at ipaddresses>
<IpInterface at FastEthernet0_1> createdTime 2010/12/21 13:15:49.236475 GMT
<IpInterface at FastEthernet0_1> iproutes <ToManyRelationship at iproutes>
<IpInterface at FastEthernet0_1> dependents <ToManyRelationship at dependents>
<IpInterface at FastEthernet0_1> type ethernetCsmacd_64
<IpInterface at FastEthernet0_10> adminStatus 1
<IpInterface at FastEthernet0_10> links <ToManyRelationship at links>
<IpInterface at FastEthernet0_10> ifindex 11
```

*Figure 48: Using zendmd in Zenoss 3 to show all attributes for all interfaces on switch, including ifindex*

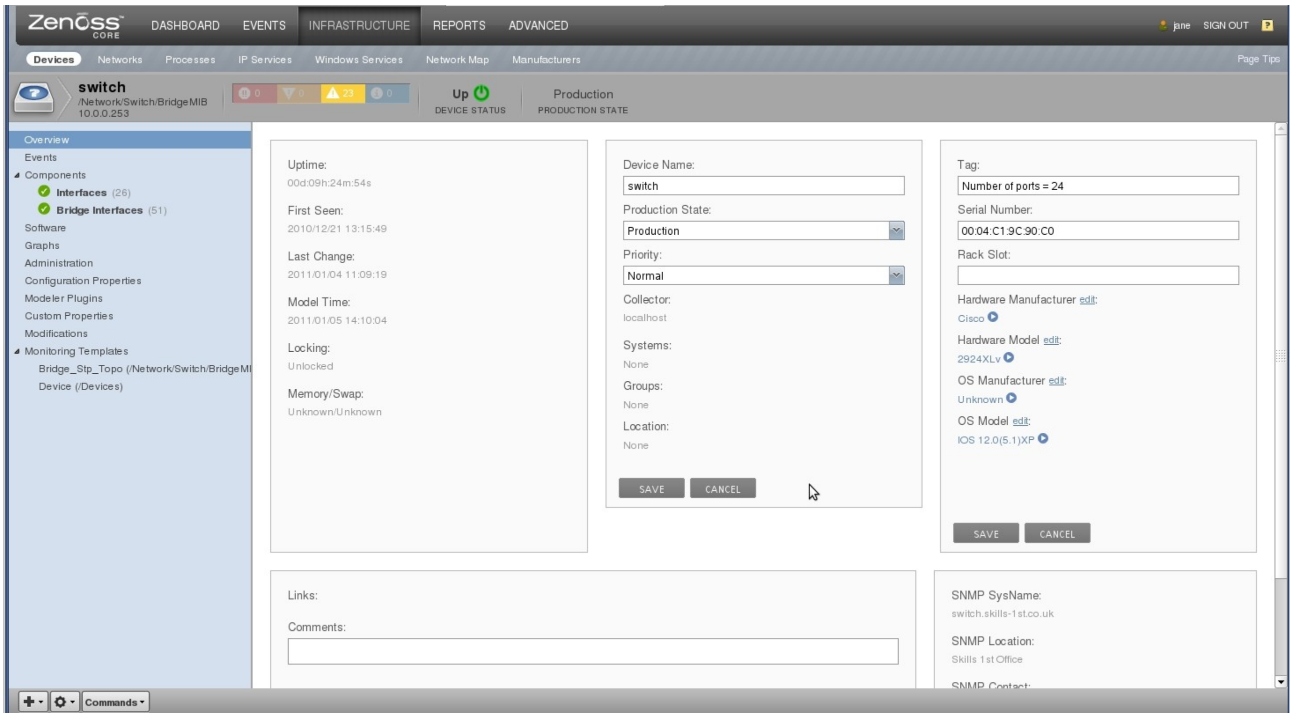
The second modeler plugin for the ZenPack is trivial in comparison but demonstrates a useful feature and a neat trick. The BridgeDeviceMib.py plugin will be activated for switch devices but will deliver device-wide information, rather than port component information.

The BRIDGE MIB delivers:

| | |
|---|---|
| .1.3.6.1.2.1.17.1.1.0 | dot1dBaseBridgeAddress |
| .1.3.6.1.2.1.17.1.2.0 | dot1dBaseNumPorts |

Now consider the standard information that is displayed for any Zenoss device on its Status page. This includes a number of standard device properties such as:

- Tag number
- Serial number
- Rack Slot



switch
/Network/Switch/BridgeMIB
10.0.0.253

Up — DEVICE STATUS
Production — PRODUCTION STATE

Uptime: 00d:09h:24m:54s
First Seen: 2010/12/21 13:15:49
Last Change: 2011/01/04 11:09:19
Model Time: 2011/01/05 14:10:04
Locking: Unlocked
Memory/Swap: Unknown/Unknown

Device Name: switch
Production State: Production
Priority: Normal
Collector: localhost
Systems: None
Groups: None
Location: None

Tag: Number of ports = 24
Serial Number: 00:04:C1:9C:90:C0
Rack Slot:
Hardware Manufacturer edit: Cisco
Hardware Model edit: 2924XLv
OS Manufacturer edit: Unknown
OS Model edit: IOS 12.0(5.1)XP

Monitoring Templates: Bridge_Stp_Topo (/Network/Switch/BridgeMI; Device (/Devices)

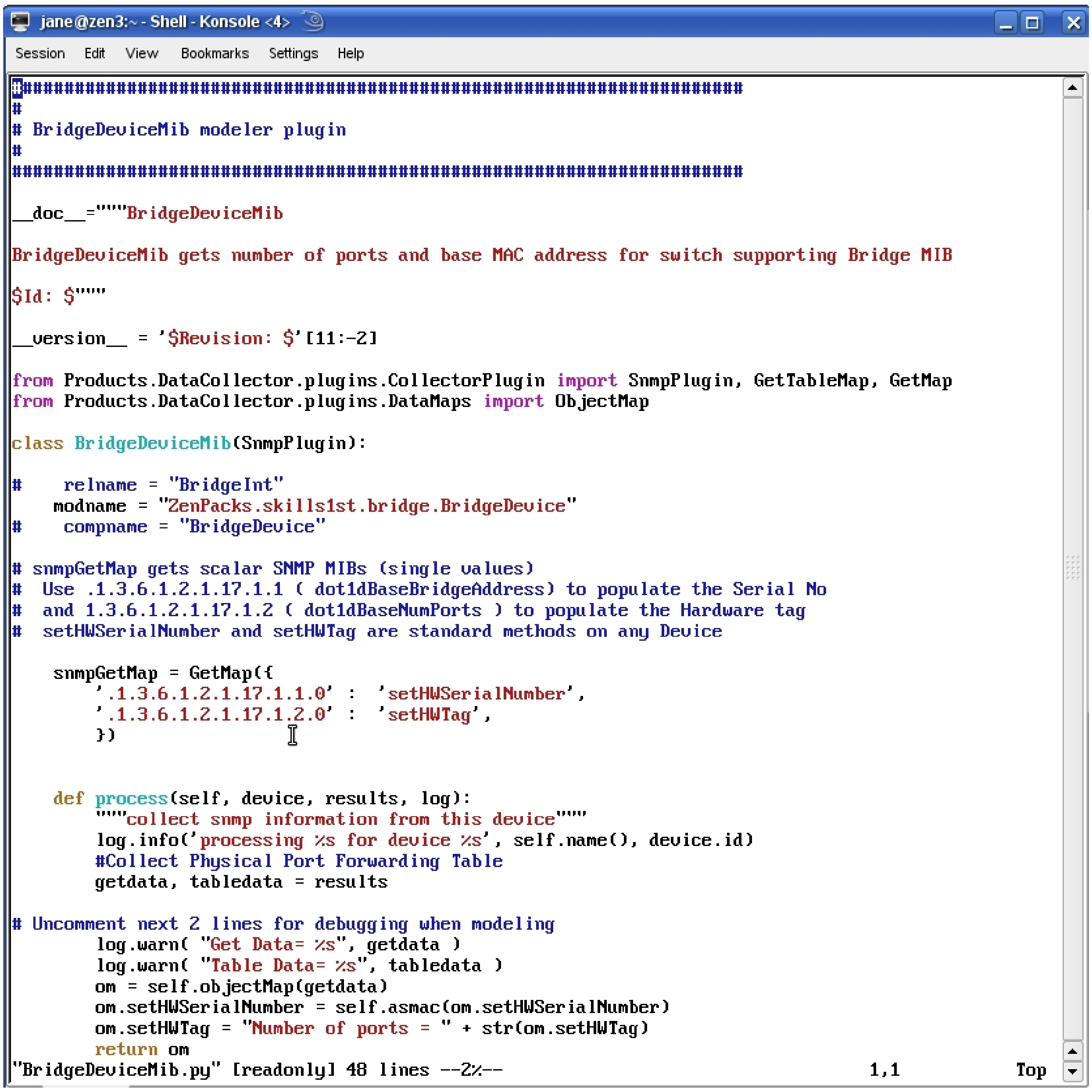
Links:
Comments:

SNMP SysName: switch.skills-1st.co.uk
SNMP Location: Skills 1st Office
SNMP Contact:

*Figure 49: Standard Overview details page for any Zenoss device*

The Tag number is not used normally for switch devices neither is the Serial number field populated; however, the Overview page for a device automatically displays data for these fields, if values exist. This is the trick that the BridgeDeviceMib plugin will use. These fields will have data mapped from the dot1dBaseBridgeAddress and dot1dBaseNumPorts OIDs described above.

So, how to get the OIDs into the relevant standard device attributes? Zenoss provides a number of *setter methods* for standard attributes, including *setHWSerialNumber* and *setHWTag* (see the Zenoss Wiki – Diving into the Device Model at http://community.zenoss.org/docs/DOC-2350 for more information on both device setters and properties) . The really useful feature that this plugin demonstrates is that SNMP data can not only be mapped to object attributes; it can also be mapped to setter methods.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
################################################################################
#
# BridgeDeviceMib modeler plugin
#
################################################################################

__doc__="""BridgeDeviceMib

BridgeDeviceMib gets number of ports and base MAC address for switch supporting Bridge MIB

$Id: $"""

__version__ = '$Revision: $'[11:-2]

from Products.DataCollector.plugins.CollectorPlugin import SnmpPlugin, GetTableMap, GetMap
from Products.DataCollector.plugins.DataMaps import ObjectMap

class BridgeDeviceMib(SnmpPlugin):

#    relname = "BridgeInt"
    modname = "ZenPacks.skills1st.bridge.BridgeDevice"
#    compname = "BridgeDevice"

# snmpGetMap gets scalar SNMP MIBs (single values)
#  Use .1.3.6.1.2.1.17.1.1 ( dot1dBaseBridgeAddress) to populate the Serial No
#  and 1.3.6.1.2.1.17.1.2 ( dot1dBaseNumPorts ) to populate the Hardware tag
#  setHWSerialNumber and setHWTag are standard methods on any Device

    snmpGetMap = GetMap({
        '.1.3.6.1.2.1.17.1.1.0' :  'setHWSerialNumber',
        '.1.3.6.1.2.1.17.1.2.0' :  'setHWTag',
        })


    def process(self, device, results, log):
        """collect snmp information from this device"""
        log.info('processing %s for device %s', self.name(), device.id)
        #Collect Physical Port Forwarding Table
        getdata, tabledata = results

# Uncomment next 2 lines for debugging when modeling
        log.warn( "Get Data= %s", getdata )
        log.warn( "Table Data= %s", tabledata )
        om = self.objectMap(getdata)
        om.setHWSerialNumber = self.asmac(om.setHWSerialNumber)
        om.setHWTag = "Number of ports = " + str(om.setHWTag)
        return om
"BridgeDeviceMib.py" [readonly] 48 lines --2%--                          1,1           Top
```

*Figure 50: BridgeDeviceMib.py modeler plugin to gather device-wide information for a switch*

This modeler populates data into the device specified by ZenPacks.skills1st.bridge.BridgeDevice ie. the device itself, not a component of the device.

Scalar data is gathered using snmpGetMap (whereas the BridgeInterfaceMib plugin used snmpGet**Table**Maps). Note that the OIDs need the trailing *.0* on the end.

```
    # snmpGetMap gets scalar SNMP MIBs (single values)
    #  Use .1.3.6.1.2.1.17.1.1 ( dot1dBaseBridgeAddress) to populate Serial No
    #  and 1.3.6.1.2.1.17.1.2 ( dot1dBaseNumPorts ) to populate Hardware tag
    #  setHWSerialNumber and setHWTag are standard methods on any Device

        snmpGetMap = GetMap({
            '.1.3.6.1.2.1.17.1.1.0' :  'setHWSerialNumber',
            '.1.3.6.1.2.1.17.1.2.0' :  'setHWTag',
            })
```

The OID values are mapped to the setHWSerialNumber and setHWTag setter methods, respectively.

In this plugin the tabledata in

```
            getdata, tabledata = results
```

will be empty (it is perfectly possible to have modeler plugins that get both scalar data and table data in the same modeler).

A single object mapping takes place, rather than a looped relationship mapping, and the data is processed slightly for readability.

```
            om = self.objectMap(getdata)
            om.setHWSerialNumber = self.asmac(om.setHWSerialNumber)
            om.setHWTag = "Number of ports = " + str(om.setHWTag)
            return om
```

The result is demonstrated in Figure 49.

Note that some other ZenPacks, particularly for Cisco devices, **will** populate the serial number field with a real serial number so you may wish to modify this part of the BridgeMib ZenPack.

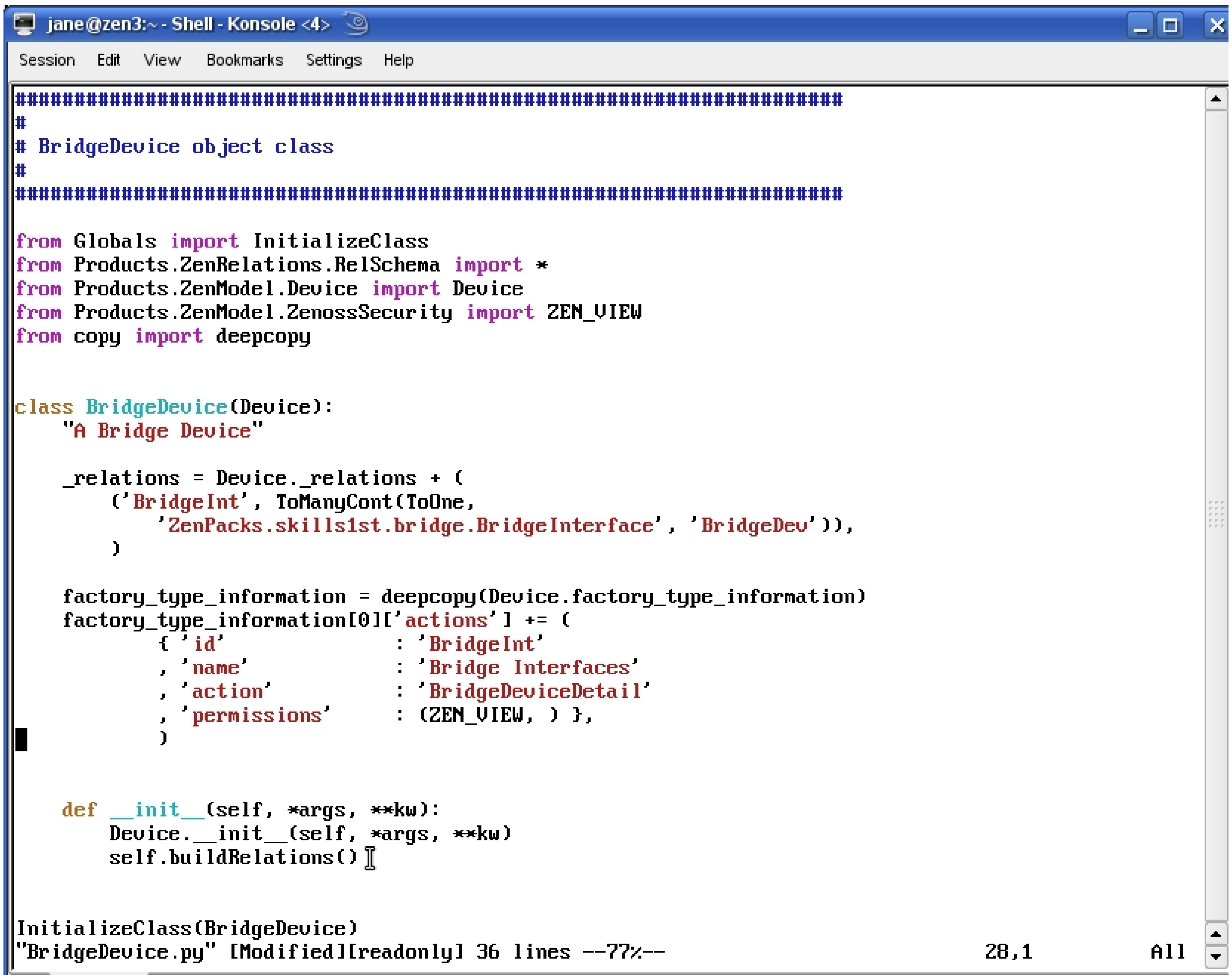
### 4.3.8 Displaying data for the ZenPack with Zenoss 2

There has been a major change between Zenoss 2 and Zenoss 3 in the way that device data is presented, and this change is still continuing. This section will first discuss how Zenoss 2 works and then address the changes required for Zenoss 3.

Zenoss 2 relies entirely on **skins** files to display data for devices, which always have a **.pt** extension.

Since the new objects are a device object (BridgeDevice) and a contained component object (BridgeInterface), for Zenoss 2, a new tab will be created for the device to augment the standard device tabs. This is preferable to adding layer 2 information to the existing OS tab as that would potentially get to be a very long page. This new “Bridge Interfaces” tab will have details of the individual ports; further, clicking on an individual port will result in a web page showing performance data for that port. So, three new elements are required.

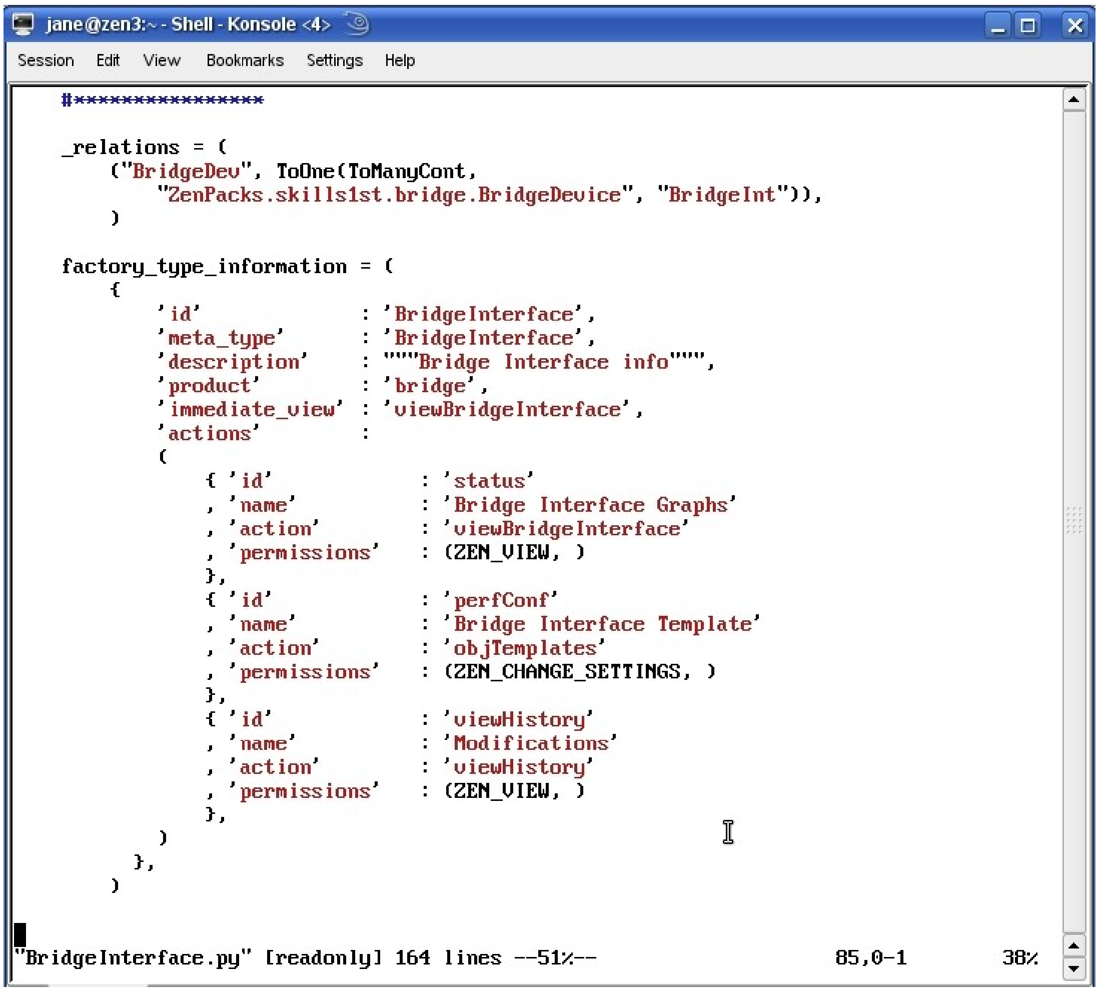
New **tabs** are created in the object class files; the **contents** of those pages are in skins files. Thus BridgeDevice.py copied all standard device tabs and added an extra tab whose label is *Bridge Interfaces* and whose skins file is called *BridgeDeviceDetail* (note that there is no .pt here but the actual file under the skins directory hierarchy must end in .pt).


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
######################################################################
#
# BridgeDevice object class
#
######################################################################

from Globals import InitializeClass
from Products.ZenRelations.RelSchema import *
from Products.ZenModel.Device import Device
from Products.ZenModel.ZenossSecurity import ZEN_VIEW
from copy import deepcopy


class BridgeDevice(Device):
    "A Bridge Device"

    _relations = Device._relations + (
        ('BridgeInt', ToManyCont(ToOne,
            'ZenPacks.skills1st.bridge.BridgeInterface', 'BridgeDev')),
        )

    factory_type_information = deepcopy(Device.factory_type_information)
    factory_type_information[0]['actions'] += (
            { 'id'              : 'BridgeInt'
            , 'name'            : 'Bridge Interfaces'
            , 'action'          : 'BridgeDeviceDetail'
            , 'permissions'     : (ZEN_VIEW, ) },
            )


    def __init__(self, *args, **kw):
        Device.__init__(self, *args, **kw)
        self.buildRelations()


InitializeClass(BridgeDevice)
"BridgeDevice.py" [Modified][readonly] 36 lines --77%--              28,1          All
```

*Figure 51: BridgeDevice.py object class file for Zenoss 2  - note the action called BridgeDeviceDetail*

Similarly, BridgeInterface.py defines three tabs, one of which is specific to the ZenPack (the viewBridgeInterface action) and two standard tabs from $ZENHOME/Products/ZenModel/skins/zenmodel (objTemplates and viewHistory). Note that BridgeInterface.py does not copy any existing tabs and that the *product* parameter must indicate the last part of the ZenPack name (*bridge* in this case).  The *immediate_view* parameter can be used to define which tab is initially opened.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
#*****************

_relations = (
    ("BridgeDev", ToOne(ToManyCont,
        "ZenPacks.skills1st.bridge.BridgeDevice", "BridgeInt")),
    )

factory_type_information = (
    {
        'id'             : 'BridgeInterface',
        'meta_type'      : 'BridgeInterface',
        'description'    : """Bridge Interface info""",
        'product'        : 'bridge',
        'immediate_view' : 'viewBridgeInterface',
        'actions'        :
        (
            { 'id'            : 'status'
            , 'name'          : 'Bridge Interface Graphs'
            , 'action'        : 'viewBridgeInterface'
            , 'permissions'   : (ZEN_VIEW, )
            },
            { 'id'            : 'perfConf'
            , 'name'          : 'Bridge Interface Template'
            , 'action'        : 'objTemplates'
            , 'permissions'   : (ZEN_CHANGE_SETTINGS, )
            },
            { 'id'            : 'viewHistory'
            , 'name'          : 'Modifications'
            , 'action'        : 'viewHistory'
            , 'permissions'   : (ZEN_VIEW, )
            },
        )
      },
    )

"BridgeInterface.py" [readonly] 164 lines --51%--                85,0-1        38%
```

*Figure 52: BridgeInterface.py object class file showing tab definitions*

Skins files defining web pages live under the *skins/ZenPacks.skills1st.bridge* subdirectory (for this ZenPack). As a general guideline, start creating skins files by looking for a sample file (on the forum, the wiki, or in the standard $ZENHOME/Products/ZenModel/skins/zenmodel directory); copy the sample and modify it to suit. Consult Chapter 13 of the Zenoss Developer's Guide 3 for lots of explanations about skins files.

If you are not familiar with the different techniques of TAL, METAL, TALES, HTML and ZPT , it can be very confusing as to what is going on!

- HyperText Markup Language (HTML) - is the most basic formatting language available on the Web, and some version of HTML is understood by every Web browser. HTML is in practice a sloppy variant of eXtensible Markup Language (XML) which divides up a page into elements (tags such as title, head or h3) and content (for example, the things that you actually care about). Common HTML tags found in Zenoss skins files include:

| | |
|---|---|
| ○ <th> | table header |
| ○ <td> | table data |

| | |
|---|---|
| ○ <tr> | table row |
| ○ <br> | break |
| ○ <block> | creates larger structures that can include other blocks |
| ○ <form> | for user input |
| ○ <input> | input directive |

- Zope Page Templates (ZPT) - are in essence HTML pages which are well-formed and have extra XML attributes (ie the bits after the element name in-between the < and > characters). The extra XML bits (attributes) are not a part of any HTML standard and are ignored by HTML editors, meaning that ZPT pages live happily with HTML. These attributes and the programming functionality that they deliver are called the Template Attribute Language (TAL). Zenoss skins files all have a *.pt* extension for Page Template.
- Template Attribute Language (TAL) - the TAL attributes allow you to add dynamic content using information from inside the Zope database (ZODB). From a Zenoss perspective, this allows you to write a query that you can use to build a table, or show different items depending on what objects or devices exist in a particular state. In other words, TAL is the Zope way of accomplishing what you would normally need to do in a CGI inside of a plain web server like Apache. It should be noted that inside TAL it is also possible to use a restricted subset of Python. The restrictions include not being able to load certain standard libraries, as well as operations like reading and writing to disk. This is done intentionally for security reasons. See http://docs.zope.org/zope2/zope2book/AppendixC.html for a Zope Page Template reference. TAL includes statements such as:

| | |
|---|---|
| ○ tal:define | define variables |
| ○ tal:condition | test conditions |
| ○ tal:content | replace the content of an element |
| ○ tal:repeat | repeat an element |
| ○ tal:replace | replace content of an element |
| ○ tal:attributes | dynamically change element attributes |

- Macro Expansion for TAL (METAL) - because TAL is hidden away inside HTML, there's no way to reuse blocks of HTML and TAL for your site just by using TAL. METAL allows page templates to define *macros* (which are essentially sub-templates that may be called by other templates) and *slots* (which may be filled by other templates). Several METAL macros are provided with Zenoss such as:

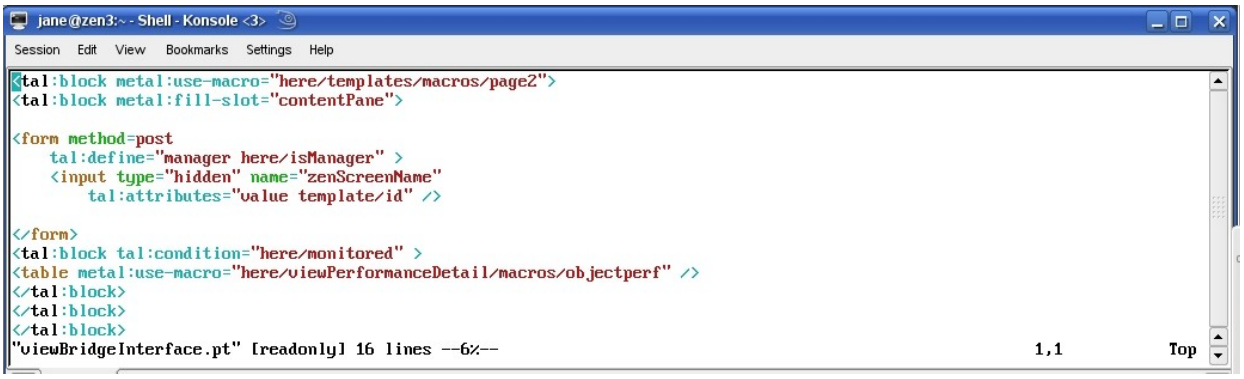
| | |
|---|---|
| ○ page1 | provides web page with breadcrumbs and content |
| ○ page2 | page 1 plus standard breadcrumbs and navigation tabs |

| | |
|---|---|
| page3 | page 1 plus standard breadcrumbs, no tabs |
| zentable | creates tables of data for display |
| navbodypagedevice | macro to support sorting, filtering, multi-pages |

- TAL Expression Syntax (TALES) - TALES allows access to the template's namespace, including useful properties such as the *here* context object. TALES accepts paths (e.g. *here / id*) which it resolves into object properties. It will attempt to resolve the final path element as a key index, a key name, an attribute, or a method. For example, if getSomething() is a method on the context, here/getSomething will return the result of that method. TALES statements are what normally provides the dynamic content for a page template, delivering data from the ZODB database.

This ZenPack has referred to two ZenPack-specific skins files; BridgeDeviceDetail.pt from BridgeDevice.py and viewBridgeInterface.pt from BridgeInterface.py.

viewBridgeInterface.pt is simple and in fact, only uses a standard METAL macro to display any performance graphs that have been customised for a port interface.


jane@zen3:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help

```
<tal:block metal:use-macro="here/templates/macros/page2">
<tal:block metal:fill-slot="contentPane">

<form method=post
    tal:define="manager here/isManager" >
    <input type="hidden" name="zenScreenName"
        tal:attributes="value template/id" />

</form>
<tal:block tal:condition="here/monitored" >
<table metal:use-macro="here/viewPerformanceDetail/macros/objectperf" />
</tal:block>
</tal:block>
</tal:block>
"viewBridgeInterface.pt" [readonly] 16 lines --6%--                    1,1          Top
```

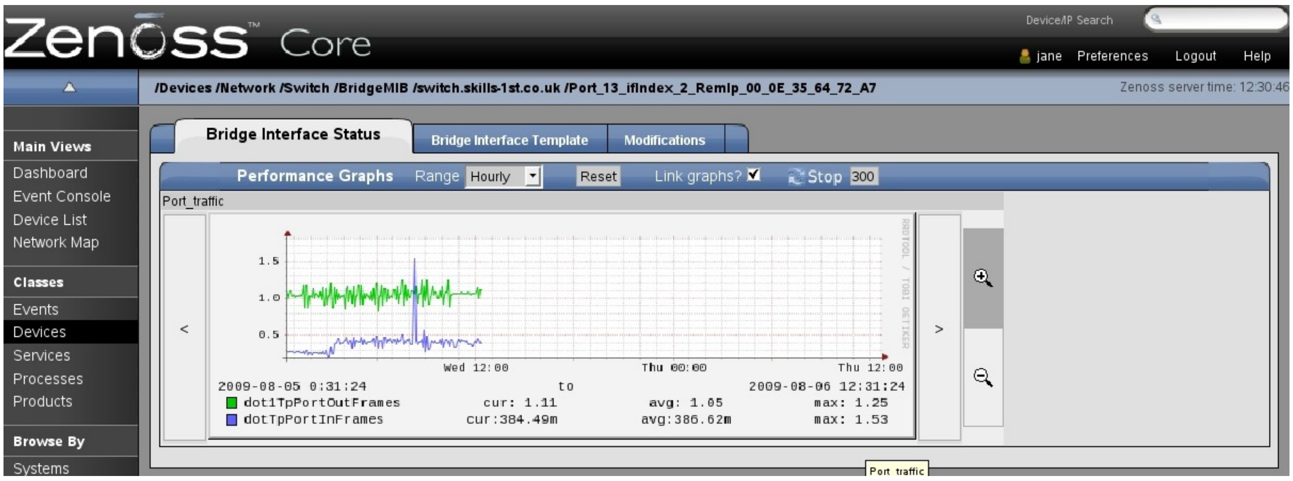
*Figure 53: viewBridgeInterface.pt skins file to display performance graphs for a port interface*

The first line calls a METAL macro to define a page with standard breadcrumbs and automatic tabs ( *here / templates / macros / page2* ).

There is a TAL condition to check that the device is being monitored.

The standard macro to display performance data for an object is called ( *here / viewPerformanceDetail / macros / objectperf* ). This macro can be found in $ZENHOME/Products/ZenModel/skins/zenmodel/viewPerformanceDetail.pt .

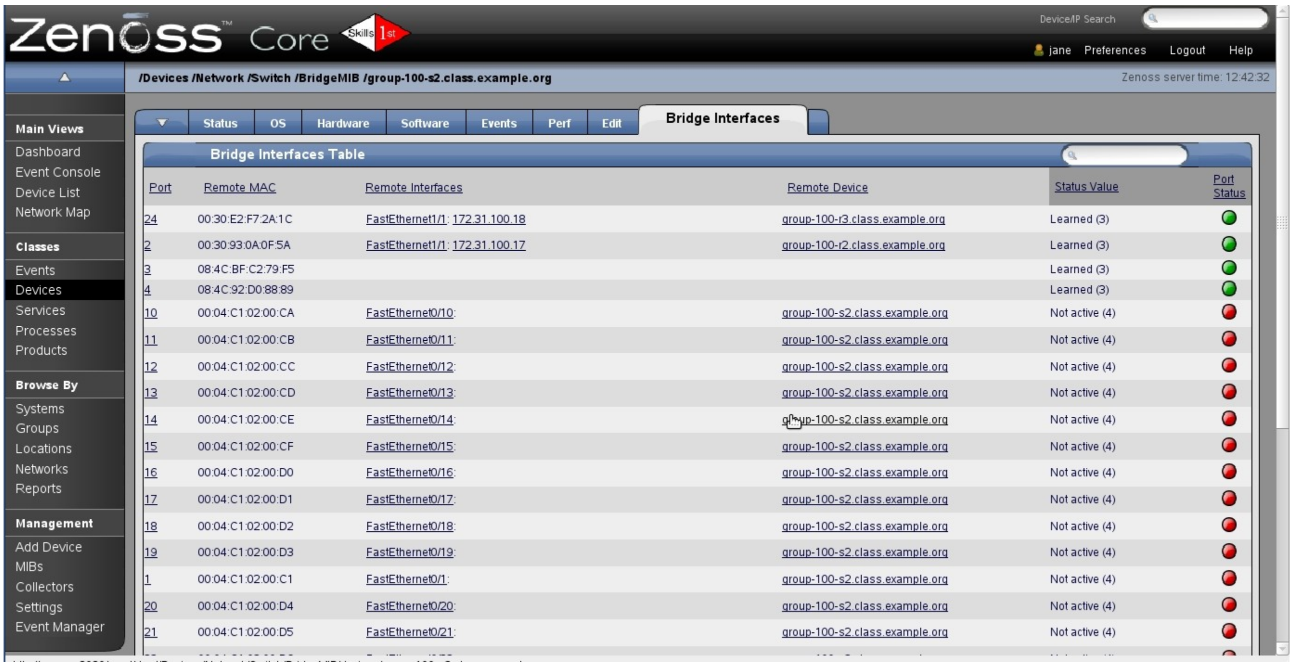
The resulting page is shown in Figure 54. At this stage, do not worry about the contents of the graph, simply that the graph is displayed with data. Performance data will be looked at in more detail later.



```
2009-08-05 0:31:24                to                2009-08-06 12:31:24
■ dot1TpPortOutFrames       cur: 1.11      avg: 1.05       max: 1.25
■ dotTpPortInFrames         cur:384.49m    avg:386.62m     max: 1.53
```

*Figure 54: Performance graph for a bridge interface in Zenoss 2*

The second skins file is more complex. It is best described alongside a screenshot of the result – see Figure 55.



| Port | Remote MAC | Remote Interfaces | Remote Device | Status Value | Port Status |
|---|---|---|---|---|---|
| 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | Learned (3) | 🟢 |
| 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | Learned (3) | 🟢 |
| 3 | 08:4C:BF:C2:79:F5 | | | Learned (3) | 🟢 |
| 4 | 08:4C:92:D0:88:89 | | | Learned (3) | 🟢 |
| 10 | 00:04:C1:02:00:CA | FastEthernet0/10: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 11 | 00:04:C1:02:00:CB | FastEthernet0/11: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 12 | 00:04:C1:02:00:CC | FastEthernet0/12: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 13 | 00:04:C1:02:00:CD | FastEthernet0/13: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 14 | 00:04:C1:02:00:CE | FastEthernet0/14: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 15 | 00:04:C1:02:00:CF | FastEthernet0/15: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 16 | 00:04:C1:02:00:D0 | FastEthernet0/16: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 17 | 00:04:C1:02:00:D1 | FastEthernet0/17: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 19 | 00:04:C1:02:00:D3 | FastEthernet0/19: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 1 | 00:04:C1:02:00:C1 | FastEthernet0/1: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 20 | 00:04:C1:02:00:D4 | FastEthernet0/20: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 21 | 00:04:C1:02:00:D5 | FastEthernet0/21: | group-100-s2.class.example.org | Not active (4) | 🔴 |

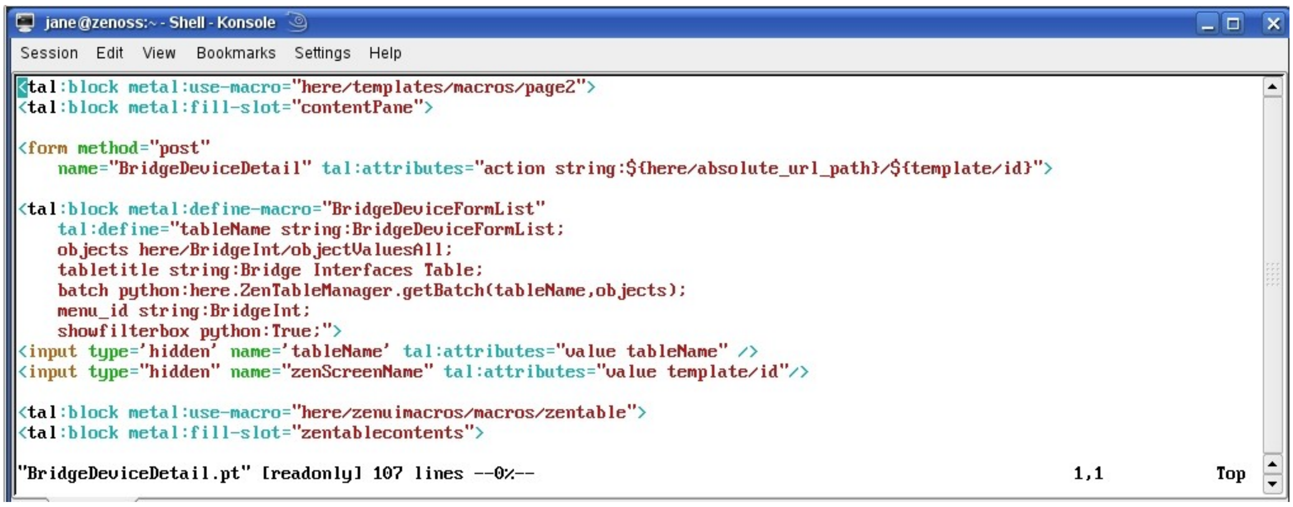
*Figure 55: Web page produced by BridgeDeviceDetail.pt for Zenoss 2*

The objective is to produce a single table with information for each port on a separate line. The readability of adjacent lines is enhanced by alternating the background colour.

Some of the port information is simply object attribute values, such as Port and RemoteAddress - the unique attributes defined in BridgeInterface.py.

Some of the information is constructed using methods of the object, again defined in BridgeInterface.py; these include Remote Interfaces and Remote Device.

The last two fields of the table present the value of the object attribute PortStatus in two different ways. Fundamentally, if the status is 3 (Learned) then it is deemed to be "active". The last field is a green bullet for an active port; otherwise the bullet is red. The previous field presents the PortStatus value but rather than just presenting the numeric value (3, 4, 5, etc), it also provides a decode for the number (*Learned* or *Not Active*).


jane@zenoss:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
<tal:block metal:use-macro="here/templates/macros/page2">
<tal:block metal:fill-slot="contentPane">

<form method="post"
    name="BridgeDeviceDetail" tal:attributes="action string:${here/absolute_url_path}/${template/id}">

<tal:block metal:define-macro="BridgeDeviceFormList"
    tal:define="tableName string:BridgeDeviceFormList;
    objects here/BridgeInt/objectValuesAll;
    tabletitle string:Bridge Interfaces Table;
    batch python:here.ZenTableManager.getBatch(tableName,objects);
    menu_id string:BridgeInt;
    showfilterbox python:True;">
<input type='hidden' name='tableName' tal:attributes="value tableName" />
<input type="hidden" name="zenScreenName" tal:attributes="value template/id"/>

<tal:block metal:use-macro="here/zenuimacros/macros/zentable">
<tal:block metal:fill-slot="zentablecontents">

"BridgeDeviceDetail.pt" [readonly] 107 lines --0%--                    1,1           Top
```

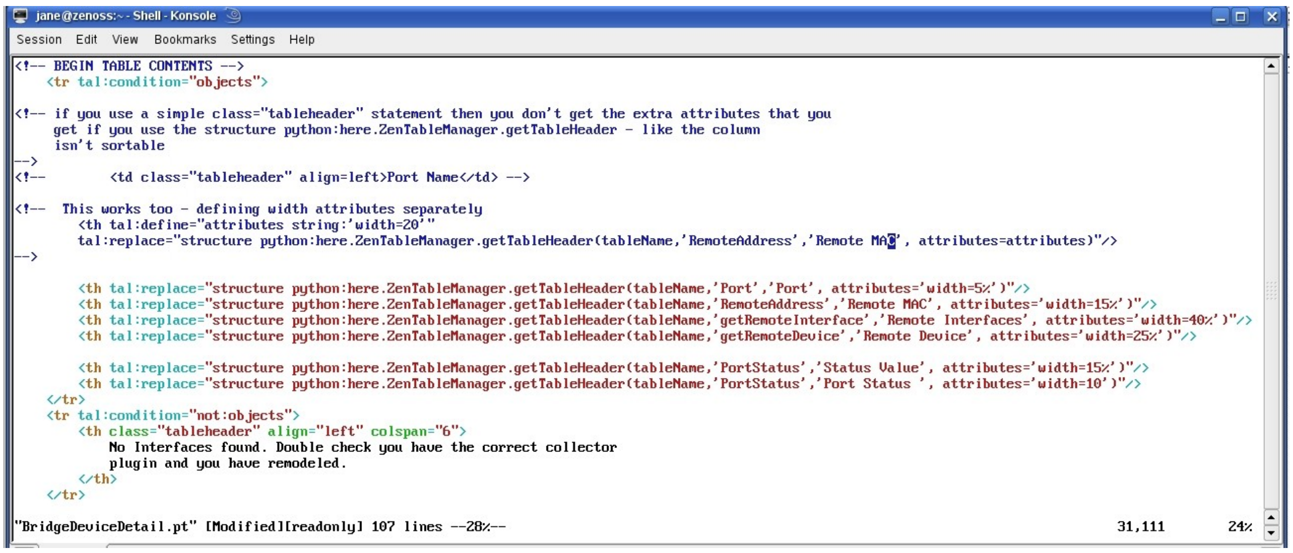
*Figure 56: BridgeDeviceDetail.pt (part 1) showing page type and BridgeDeviceFormList macro*

The first line of BridgeDeviceDetail.pt uses the page2 macro again for a page with breadcrumbs and tabs.

The *BridgeDeviceFormList* macro is defined to get all the objects from the device's BridgeInt relationship ( *here/BridgeInt/objectValuesAll* ) and supply them in a table. Since there may be many interfaces, the filter box (at the top right of the GUI page) should be enabled ( *showfilterbox python:True* ).

```
<tal:block metal:define-macro="BridgeDeviceFormList"
    tal:define="tableName string:BridgeDeviceFormList;
    objects here/BridgeInt/objectValuesAll;
    tabletitle string:Bridge Interfaces Table;
    batch python:here.ZenTableManager.getBatch(tableName,objects);
    menu_id string:BridgeInt;
    showfilterbox python:True;">
```

The second part of the skins file defines the table headers with their layout.


jane@zenoss:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
<!-- BEGIN TABLE CONTENTS -->
    <tr tal:condition="objects">

<!-- if you use a simple class="tableheader" statement then you don't get the extra attributes that you
     get if you use the structure python:here.ZenTableManager.getTableHeader - like the column
     isn't sortable
-->
<!--        <td class="tableheader" align=left>Port Name</td> -->

<!--  This works too - defining width attributes separately
        <th tal:define="attributes string:'width=20'"
        tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'RemoteAddress','Remote MAC', attributes=attributes)"/>
-->

        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'Port','Port', attributes='width=5%')"/>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'RemoteAddress','Remote MAC', attributes='width=15%')"/>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'getRemoteInterface','Remote Interfaces', attributes='width=40%')"/>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'getRemoteDevice','Remote Device', attributes='width=25%')"/>

        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'PortStatus','Status Value', attributes='width=15%')"/>
        <th tal:replace="structure python:here.ZenTableManager.getTableHeader(tableName,'PortStatus','Port Status ', attributes='width=10')"/>
    </tr>
    <tr tal:condition="not:objects">
        <th class="tableheader" align="left" colspan="6">
            No Interfaces found. Double check you have the correct collector
            plugin and you have remodeled.
        </th>
    </tr>

"BridgeDeviceDetail.pt" [Modified][readonly] 107 lines --28%--                    31,111        24%
```

*Figure 57: BridgeDeviceDetail.pt (part 2) showing table headers layout*

The line:

```
<tr tal:condition="objects">
```

starts the definition of the table row ( <tr matched by the closing </tr> 6 lines from the end of Figure 57), and uses a TAL statement to ensure that the variable *objects* was actually populated from *here/BridgeInt/objectValuesAll* in the earlier section. If *objects* is null then the remainder of the <tr> row definition will be ignored..

There are lots of permutations for structuring header and data lines of a table. The comments in Figure 57 explain some of the consequences. A flexible way is to use lines like the following:

```
<th tal:replace="structure
python:here.ZenTableManager.getTableHeader(tableName,'RemoteAddress',
'Remote MAC', attributes='width=15%')"/>
```

The table header (< th ... />) uses a TAL replace statement to use Python to get values from the table defined, accessing the object attribute value (RemoteAddress) and using the string 'Remote MAC' as the column header, allowing 15 characters width.

Another alternative would be to use a table data (<td ... /> ) HTML tag but this seems to result in a table where columns are not sortable:

```
<td class="tableheader" align=left>Port Name</td>
```

If the attributes are more complex or extensive, they can be declared separately:

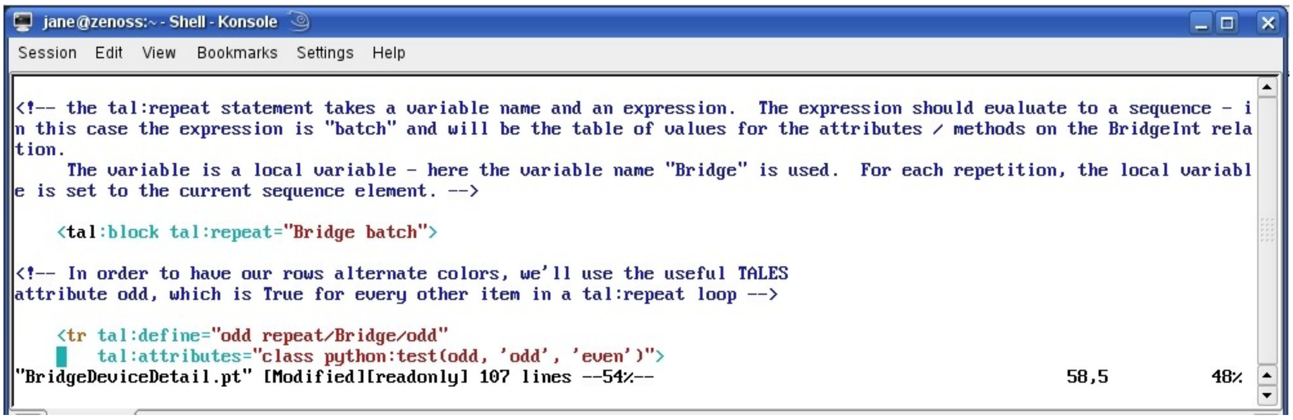
```
<th tal:define="attributes string:'width=20'"
   tal:replace="structure
python:here.ZenTableManager.getTableHeader(tableName,'RemoteAddress',
'Remote MAC', attributes=attributes)"/>
```

If the *objects* variable is null then a warning message is displayed:

```
    <tr tal:condition="not:objects">
        <th class="tableheader" align="left" colspan="6">
```

```
            No Interfaces found. Double check you have the correct
collector plugin and you have remodeled.
        </th>
    </tr>
```

Next, a block is set up that will repeatedly output one row of the table for each port, with alternate lines having a different background.


jane@zenoss:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
<!-- the tal:repeat statement takes a variable name and an expression.  The expression should evaluate to a sequence - i
n this case the expression is "batch" and will be the table of values for the attributes / methods on the BridgeInt rela
tion.
     The variable is a local variable - here the variable name "Bridge" is used.  For each repetition, the local variabl
e is set to the current sequence element. -->

    <tal:block tal:repeat="Bridge batch">

<!-- In order to have our rows alternate colors, we'll use the useful TALES
attribute odd, which is True for every other item in a tal:repeat loop -->

    <tr tal:define="odd repeat/Bridge/odd"
        tal:attributes="class python:test(odd, 'odd', 'even')">
"BridgeDeviceDetail.pt" [Modified][readonly] 107 lines --54%--                          58,5          48%
```

*Figure 58: BridgeDeviceDetail.pt (part 3) showing the controls for the data rows of the port table*

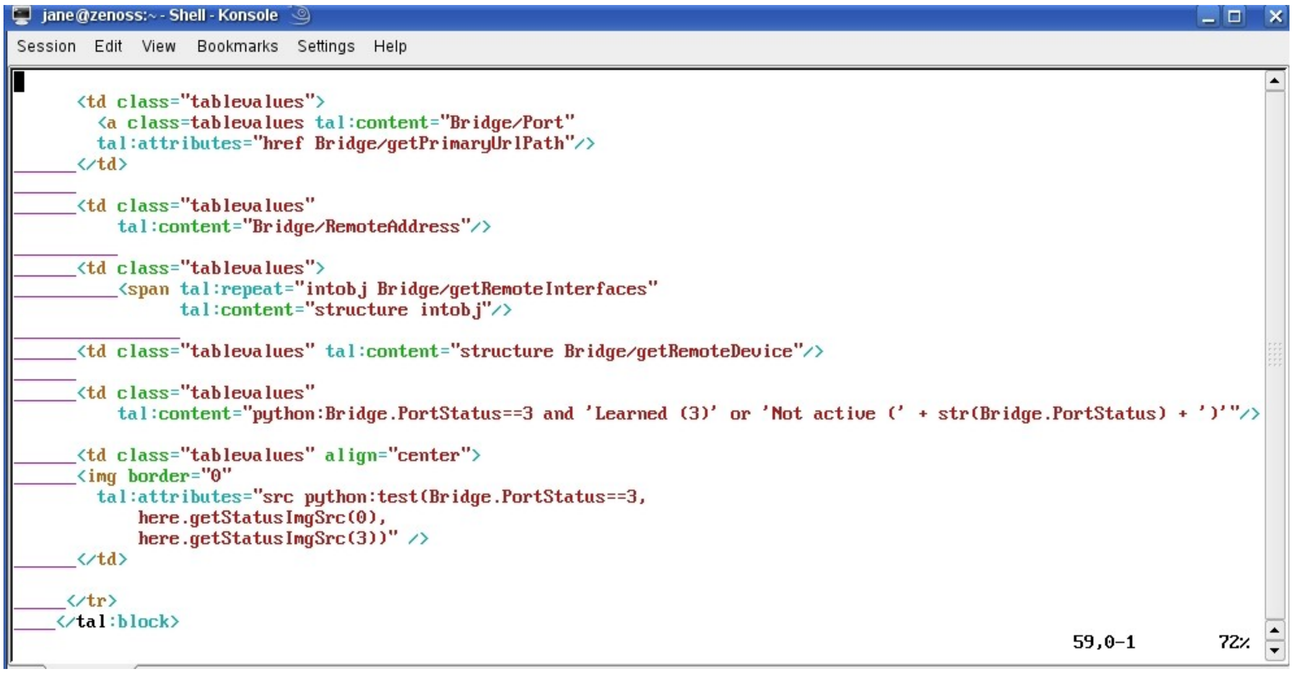
The *tal:repeat* statement takes a variable name and an expression. The expression should evaluate to a sequence - in this case the expression is *batch* (defined earlier in the BridgeDeviceFormList macro) and will be the table of values for the attributes / methods on the BridgeInt relationship. The variable is a local variable - here the variable name *Bridge* is used. For each repetition, the local variable is set to the current sequence element.

```
<tal:block tal:repeat="Bridge batch">
```

A standard TALES attribute, *odd*, can be used which evaluates to True for every other item in a tal:repeat loop. It provides different background colours for alternate lines. This code fraction also shows the start of the table row definition ( <tr ).

```
<tr tal:define="odd repeat/Bridge/odd"
        tal:attributes="class python:test(odd, 'odd', 'even')">
```

The next section provides the data values for a row of the table.



jane@zenoss:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
      <td class="tablevalues">
        <a class=tablevalues tal:content="Bridge/Port"
        tal:attributes="href Bridge/getPrimaryUrlPath"/>
______</td>
______
______<td class="tablevalues"
          tal:content="Bridge/RemoteAddress"/>
__________
______<td class="tablevalues">
__________<span tal:repeat="intobj Bridge/getRemoteInterfaces"
                tal:content="structure intobj"/>
________________
______<td class="tablevalues" tal:content="structure Bridge/getRemoteDevice"/>
______
______<td class="tablevalues"
          tal:content="python:Bridge.PortStatus==3 and 'Learned (3)' or 'Not active (' + str(Bridge.PortStatus) + ')'"/>

______<td class="tablevalues" align="center">
______<img border="0"
        tal:attributes="src python:test(Bridge.PortStatus==3,
            here.getStatusImgSrc(0),
            here.getStatusImgSrc(3))" />
______</td>

_____</tr>
____</tal:block>
                                                                      59,0-1        72%
```

*Figure 59: BridgeDeviceDetail.pt (part 4) showing the data values for the port table*

Each table data ( <td> ) tag uses a tal:content statement to reference either an attribute or a method on the BridgeInterface object to deliver a data value. Remember that *Bridge* is the local variable that takes the next set of values from the port table, each time round the tal:repeat loop. Either object attributes or object methods can be used to provide the table data values.

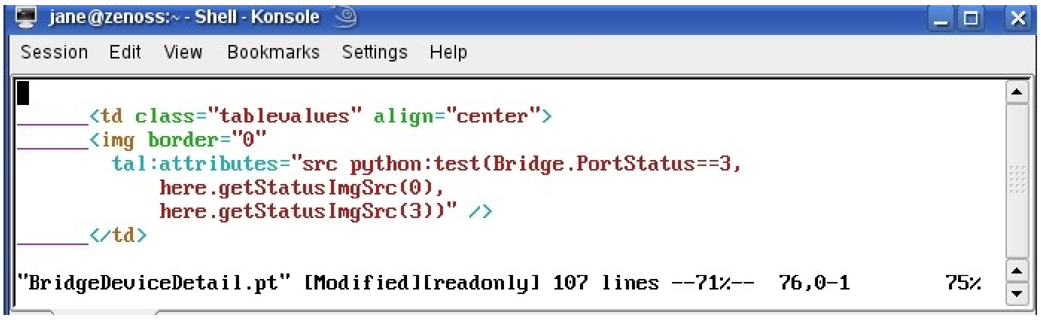
```
<td class="tablevalues">
        <a class=tablevalues tal:content="Bridge/Port"
        tal:attributes="href Bridge/getPrimaryUrlPath"/>
      </td>
```

The first PortStatus data value, rather than simply showing the numeric value from the object, will also “translate” the numeric value into a more useful human representation. This uses a Python test:

```
<td class="tablevalues"
           tal:content="python:Bridge.PortStatus==3 and 'Learned (3)' or
            'Not active (' + str(Bridge.PortStatus) + ')'"/>
```

If the PortStatus of this switch port is 3 then the output will be the string 'Learned (3)'; otherwise the output will be the string 'Not active' concatenated with the string representation of the value of PortStatus, concatenated with a closing ')' .

The final data column in the table of data is a red or green bullet representing either an active port (with PortStatus = 3) or a non-active port.

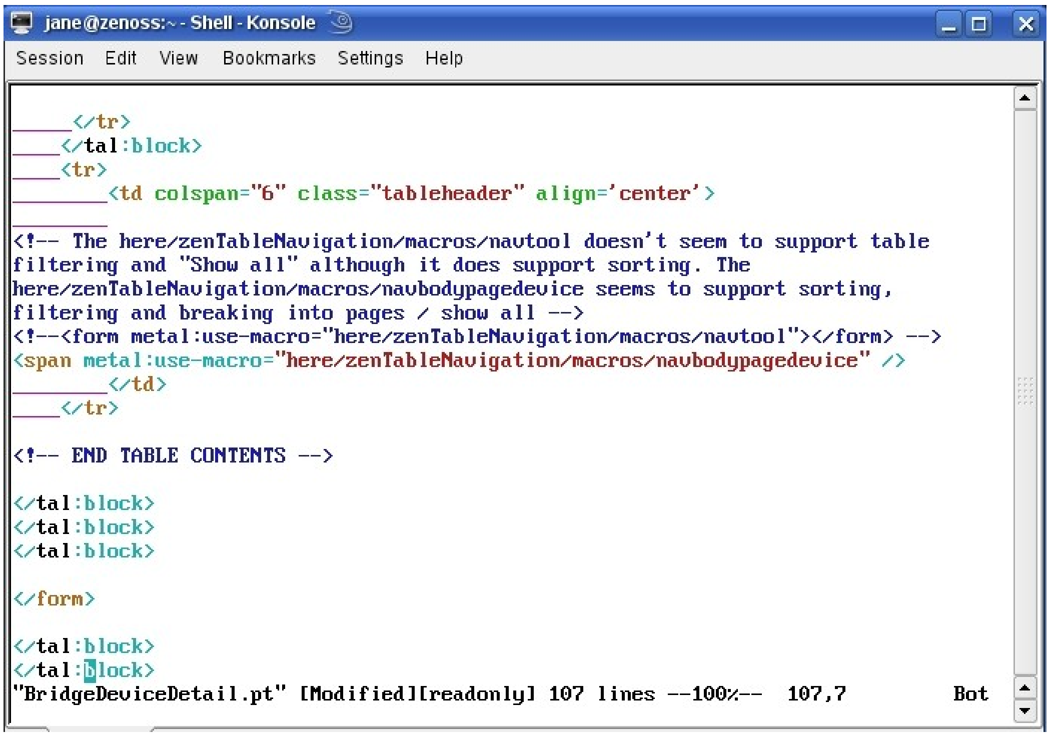

jane@zenoss:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
______<td class="tablevalues" align="center">
______<img border="0"
        tal:attributes="src python:test(Bridge.PortStatus==3,
            here.getStatusImgSrc(0),
            here.getStatusImgSrc(3))" />
______</td>

"BridgeDeviceDetail.pt" [Modified][readonly] 107 lines --71%--   76,0-1        75%
```

*Figure 60: BridgeDeviceDetail.pt (part 5) showing code to produce coloured bullets to represent PortStatus*

This is code used in several places in the standard Zenoss skins files. Again, it uses a Python test to evaluate PortStatus and then uses here.getStatusImgSrc(0) to represent a green bullet and here.getStatusImgSrc(3) for a red bullet.

The remainder of BridgeDeviceDetail.pt has the closing table row tag and the closing block tag for the data rows. The standard METAL macro *navbodypagedevice* is called to ensure that the table can be searched, the columns can be ordered and large numbers of rows will correctly be split into pages. Note that the earlier *navtool* macro does not seem to implement filtering and paging correctly. The last few lines of the file are the closing tags for blocks and the overall form.


jane@zenoss:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
_____</tr>
____</tal:block>
____<tr>
________<td colspan="6" class="tableheader" align='center'>
________
<!-- The here/zenTableNavigation/macros/navtool doesn't seem to support table
filtering and "Show all" although it does support sorting. The
here/zenTableNavigation/macros/navbodypagedevice seems to support sorting,
filtering and breaking into pages / show all -->
<!--<form metal:use-macro="here/zenTableNavigation/macros/navtool"></form> -->
<span metal:use-macro="here/zenTableNavigation/macros/navbodypagedevice" />
________</td>
____</tr>

<!-- END TABLE CONTENTS -->

</tal:block>
</tal:block>
</tal:block>

</form>

</tal:block>
</tal:block>
"BridgeDeviceDetail.pt" [Modified][readonly] 107 lines --100%--   107,7        Bot
```

*Figure 61: BridgeDeviceDetail.pt (part 6) with closing tags and the navbodypagedevice macro call*

### 4.3.9 Displaying data for the ZenPack with Zenoss 3

Chapter 14 of the Zenoss 3 Developer's Guide provides some information about converting ZenPacks from Zenoss 2 to Zenoss 3 but it is extremely terse. It depends on understanding the difference between a **re-skinned** page and a **redesigned** page.

#### 4.3.9.1 What needs changing between Zenoss 2 and Zenoss 3?

My understanding is that a **redesigned** page is one whose new page description is written in JavaScript rather than in Page Template language, although the original, redundant pt files still exist in directories such as $ZENHOME/Products/ZenModel/skins/zenmodel (like deviceOsDetail.pt). Many of the Page Template (.pt) files from Zenoss 2 have been modified for Zenoss 3 – hence **re-skinned** pages.

Chapter 14 documents the following pages as redesigned:

| **Zenoss 2** | **Zenoss 3** |
|---|---|
| Devices List, Devices Class, Systems, Groups, Locations | Devices |
| Device Status | Device Detail |
| Event Console | Event Console |
| Services Class | IP Services, Windows Services |
| Templates | Monitoring Templates |
| Networks | Networks |
| Processes Class | Processes |
| Reports | Reports |

*Table 4.3: Redesigned pages in Zenoss 3*

The conversion guide states that no changes are needed to a ZenPack if it:

- Adds tabs
- Adds dialogs to a re-skinned page

However, you must modify your ZenPack for compatibility if it:

- Overrides the page template of a redesigned page
- Adds a page-level dialog
- Includes custom data sources and thresholds

If your ZenPack didn't provide any skins files, then you're fine. If you simply provided new modeler plugins and device types that didn't have new components, then there are no changes needed.

If it added tabs for custom **non-component** data, then you're fine.

I have not found many useful re-skinned pages to which one would want to add a dialog (such as a custom Add, Delete or Edit – the sort of option that typically came from the table dropdown menus in Zenoss 2).

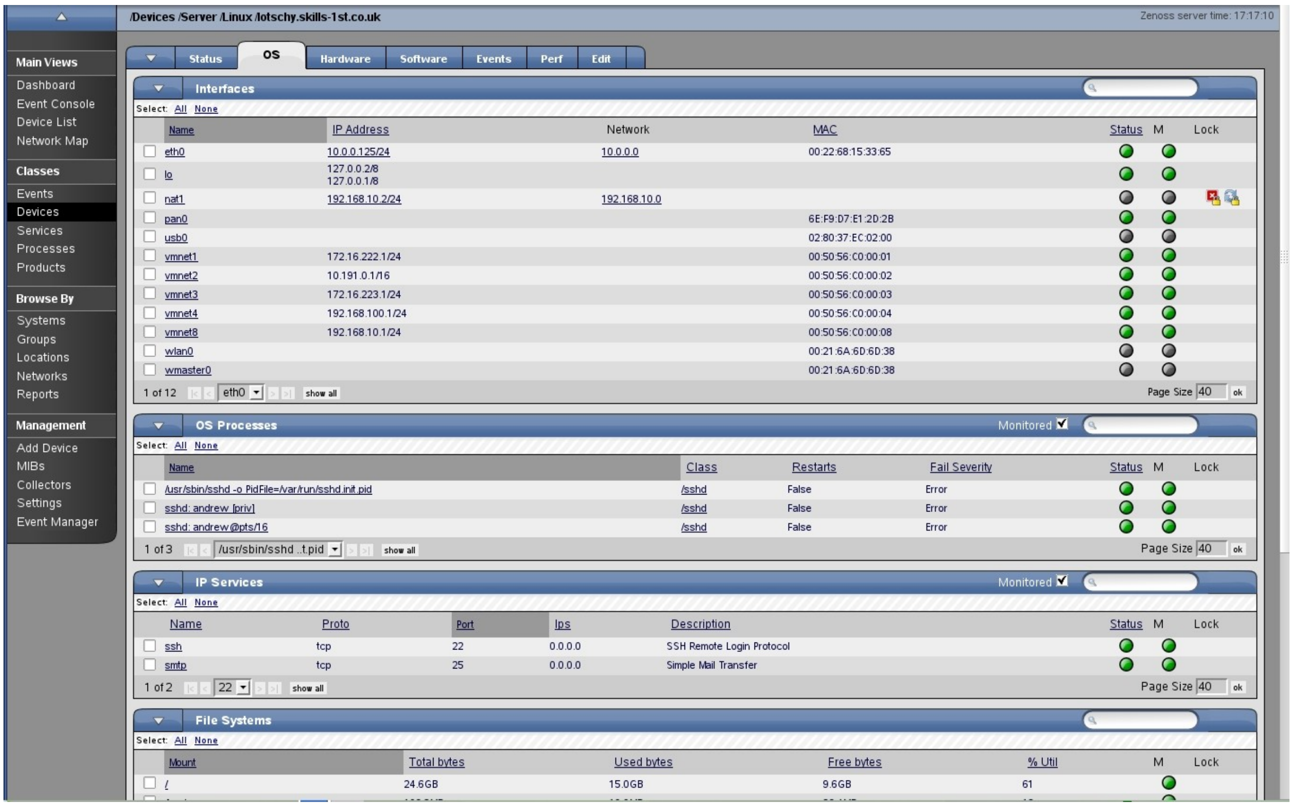
What this really comes down to is that:

- Any ZenPack that wishes to display different component details or create new components and display data for them, you need to write some JavaScript.
- If you want to add your own items to the Action icon menu or Commands menu or create new Add <item> options from the "+" icon menu, then you need to write some JavaScript.
- If you have created your own data sources then you need to write some JavaScript.

The Zenoss 2 skins files for devices and infrastructure can typically be found under **$ZENHOME/Products/ZenModel/skins/zenmodel** (deviceList.pt, deviceListMacro.pt, deviceStatus.pt, deviceOsDetail.pt, etc) with page template files for events under **$ZENHOME/Products/ZenEvents/skins/zenevents** (viewEvents.pt, ....).

Zenoss 3 stores many of its new JavaScript files under $ZENHOME/ZenUI3, which did exist in Zenoss 2, but has been greatly extended in version 3. The main subdirectory for web page layouts is **$ZENHOME/ZenUI3/browser/resources/js/zenoss** with files like DevicePanels.js to show the list of devices, DeviceOverviewPanel.js for a device overview template, devdetails.js for the main layout of device details, including the left-hand menu, and the options from the Action icon.

Perhaps the greatest change in GUI design between Zenoss 2 and Zenoss 3 is the way that components of a device are displayed. With Zenoss 2, the OS tab showed the components such as interfaces, filesystems and processes.



/Devices /Server /Linux /lotschy.skills-1st.co.uk

Zenoss server time: 17:17:10

**Interfaces**

Select: All None

| Name | IP Address | Network | MAC | Status | M | Lock |
|---|---|---|---|---|---|---|
| eth0 | 10.0.0.125/24 | 10.0.0.0 | 00:22:68:15:33:65 | ● | ● | |
| lo | 127.0.0.2/8<br>127.0.0.1/8 | | | ● | ● | |
| nat1 | 192.168.10.2/24 | 192.168.10.0 | | ● | ● | |
| pan0 | | | 6E:F9:D7:E1:2D:2B | ● | ● | |
| usb0 | | | 02:80:37:EC:02:00 | ● | ● | |
| vmnet1 | 172.16.222.1/24 | | 00:50:56:C0:00:01 | ● | ● | |
| vmnet2 | 10.191.0.1/16 | | 00:50:56:C0:00:02 | ● | ● | |
| vmnet3 | 172.16.223.1/24 | | 00:50:56:C0:00:03 | ● | ● | |
| vmnet4 | 192.168.100.1/24 | | 00:50:56:C0:00:04 | ● | ● | |
| vmnet8 | 192.168.10.1/24 | | 00:50:56:C0:00:08 | ● | ● | |
| wlan0 | | | 00:21:6A:6D:6D:38 | ● | ● | |
| wmaster0 | | | 00:21:6A:6D:6D:38 | ● | ● | |

1 of 12 eth0 show all Page Size 40 ok

**OS Processes** Monitored ✓

Select: All None

| Name | Class | Restarts | Fail Severity | Status | M | Lock |
|---|---|---|---|---|---|---|
| /usr/sbin/sshd -o PidFile=/var/run/sshd.init.pid | /sshd | False | Error | ● | ● | |
| sshd: andrew [priv] | /sshd | False | Error | ● | ● | |
| sshd: andrew@pts/16 | /sshd | False | Error | ● | ● | |

1 of 3 /usr/sbin/sshd ..t.pid show all Page Size 40 ok

**IP Services** Monitored ✓

Select: All None

| Name | Proto | Port | Ips | Description | Status | M | Lock |
|---|---|---|---|---|---|---|---|
| ssh | tcp | 22 | 0.0.0.0 | SSH Remote Login Protocol | ● | ● | |
| smtp | tcp | 25 | 0.0.0.0 | Simple Mail Transfer | ● | ● | |

1 of 2 22 show all Page Size 40 ok

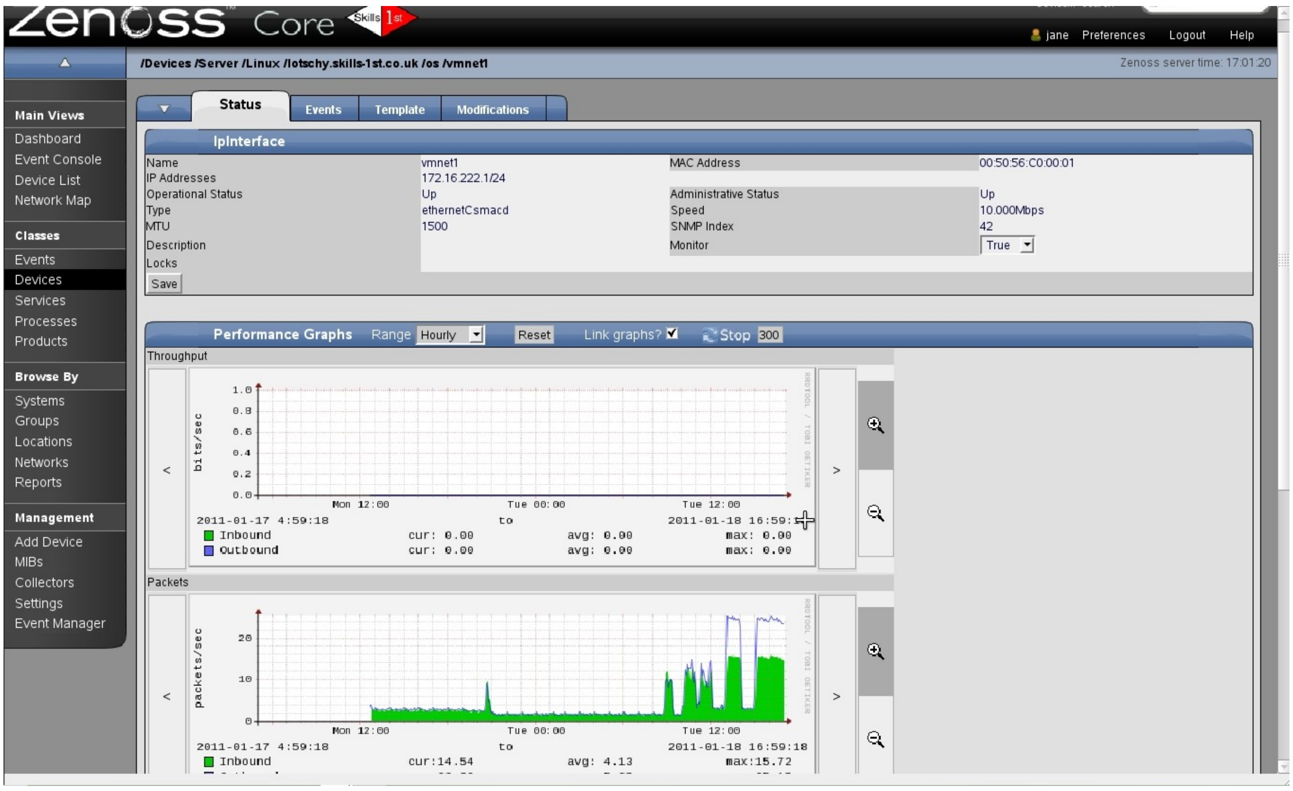
**File Systems**

Select: All None

| Mount | Total bytes | Used bytes | Free bytes | % Util | M | Lock |
|---|---|---|---|---|---|---|
| / | 24.6GB | 15.0GB | 9.6GB | 61 | ● | |

*Figure 62: OS tab for server device in Zenoss 2*

Clicking on an individual  interface / filesystem / process resulted in a window that contained both detailed information (such as interface name, type, MTU, OperationalStatus, etc) and a set of performance graphs for the relevant interface, as shown in Figure 63.



/Devices /Server /Linux /lotschy.skills-1st.co.uk /os /vmnet1

Zenoss server time: 17:01:20

IpInterface

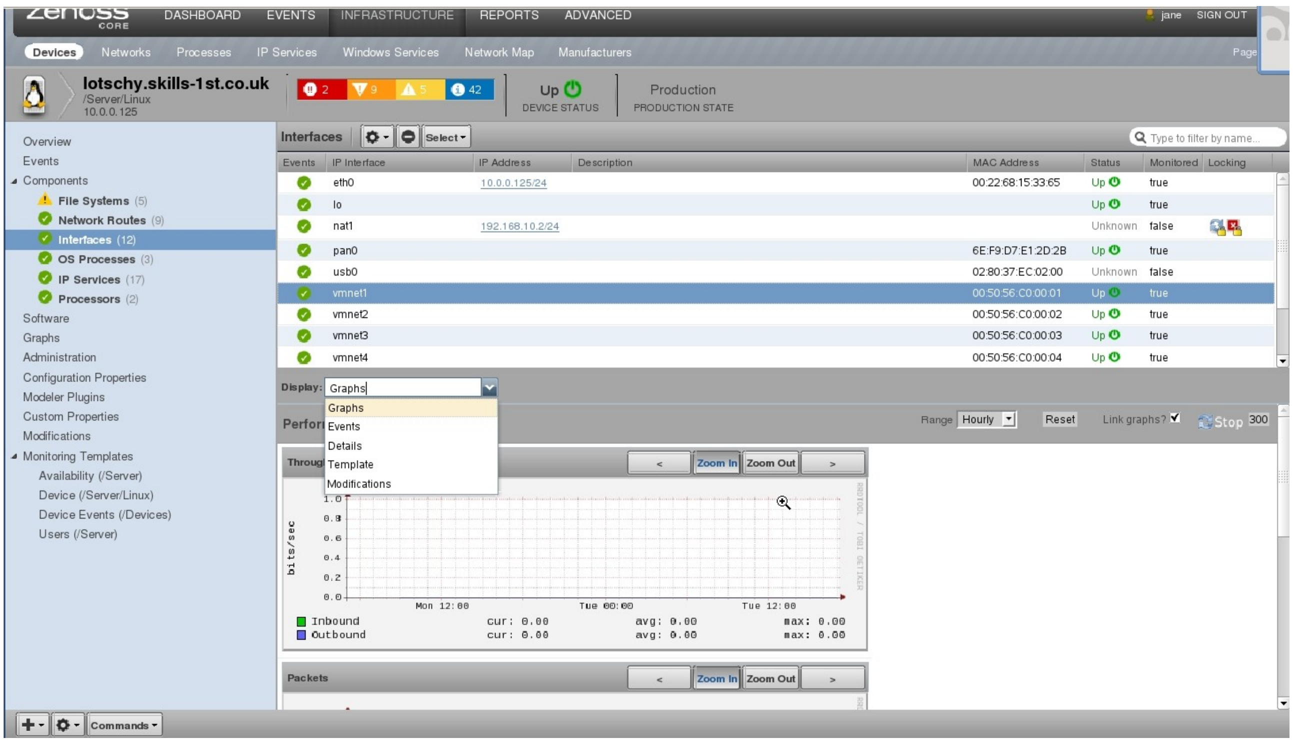
| | | | |
|---|---|---|---|
| Name | vmnet1 | MAC Address | 00:50:56:C0:00:01 |
| IP Addresses | 172.16.222.1/24 | | |
| Operational Status | Up | Administrative Status | Up |
| Type | ethernetCsmacd | Speed | 10.000Mbps |
| MTU | 1500 | SNMP Index | 42 |
| Description | | Monitor | True |
| Locks | | | |

```
2011-01-17 4:59:18              to            2011-01-18 16:59:18
Inbound              cur: 0.00        avg: 0.00        max: 0.00
Outbound             cur: 0.00        avg: 0.00        max: 0.00
```

```
2011-01-17 4:59:18              to            2011-01-18 16:59:18
Inbound              cur:14.54        avg: 4.13        max:15.72
```

*Figure 63: Interface component details and graphs for device in Zenoss 2*

With Zenoss 3, the OS tab has gone and the left-hand menu from a device's detailed page, includes a component submenu listing each component type. Selecting the component type offers a list of instances (for example the interfaces); to see the detailed information or the performance graphs for an interface, the *Display* dropdown menu is provided in the middle of the panel, shown in Figure 64.



lotschy.skills-1st.co.uk
/Server/Linux
10.0.0.125

| Events | IP Interface | IP Address | Description | MAC Address | Status | Monitored | Locking |
|---|---|---|---|---|---|---|---|
| ✓ | eth0 | 10.0.0.125/24 | | 00:22:68:15:33:65 | Up | true | |
| ✓ | lo | | | | Up | true | |
| ✓ | nat1 | 192.168.10.2/24 | | | Unknown | false | |
| ✓ | pan0 | | | 6E:F9:D7:E1:2D:2B | Up | true | |
| ✓ | usb0 | | | 02:80:37:EC:02:00 | Unknown | false | |
| ✓ | vmnet1 | | | 00:50:56:C0:00:01 | Up | true | |
| ✓ | vmnet2 | | | 00:50:56:C0:00:02 | Up | true | |
| ✓ | vmnet3 | | | 00:50:56:C0:00:03 | Up | true | |
| ✓ | vmnet4 | | | 00:50:56:C0:00:04 | Up | true | |

Display: Graphs

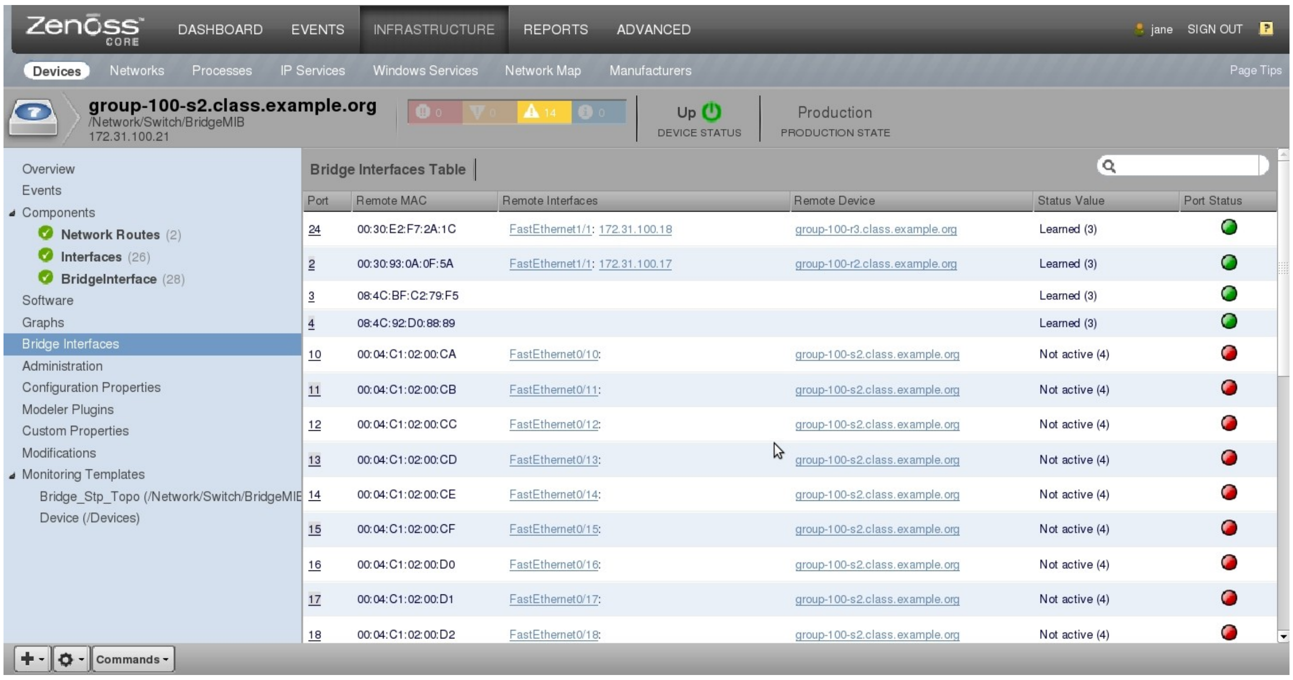
*Figure 64: Interface component details in Zenoss 3 with dropdown Display menu*

Note also in Figure 64 that the *Components* submenu includes *Processors* which used to be under the *Hardware* tab for Zenoss 2. The other element that the *Hardware* tab displayed was **Memory**, which is now on the *Overview* page for a device, in Zenoss 3.

#### 4.3.9.2 BridgeMIB ZenPack without any changes to presentation code

So, for the BridgeMIB ZenPack, in Zenoss 2 a new tab was added for *Bridge Interfaces*. If we had chosen to expand the OS tab with the layer 2 bridge interface information, then we would break the rule of "overriding the page template of a redesigned page" as this has moved to the new component detail page, but simply a "new tab" should not require a change.

However, the *Bridge Interfaces* tab displays data related to a new component of a device. We pass the "no-changes" test in that a new tab was added and there are no page dialogs or custom data sources in the ZenPack, but in practise, a lot of functionality is lost as the detailed interface information and the performance graphs are lost. Figure 65 shows the standalone left-hand *Bridge Interfaces* menu which is formatted by the original page template file, BridgeDeviceDetail.pt in the ZenPack's skins subdirectory. **Note** that there is also a *BridgeInterface* menu as a submenu of *Components*.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

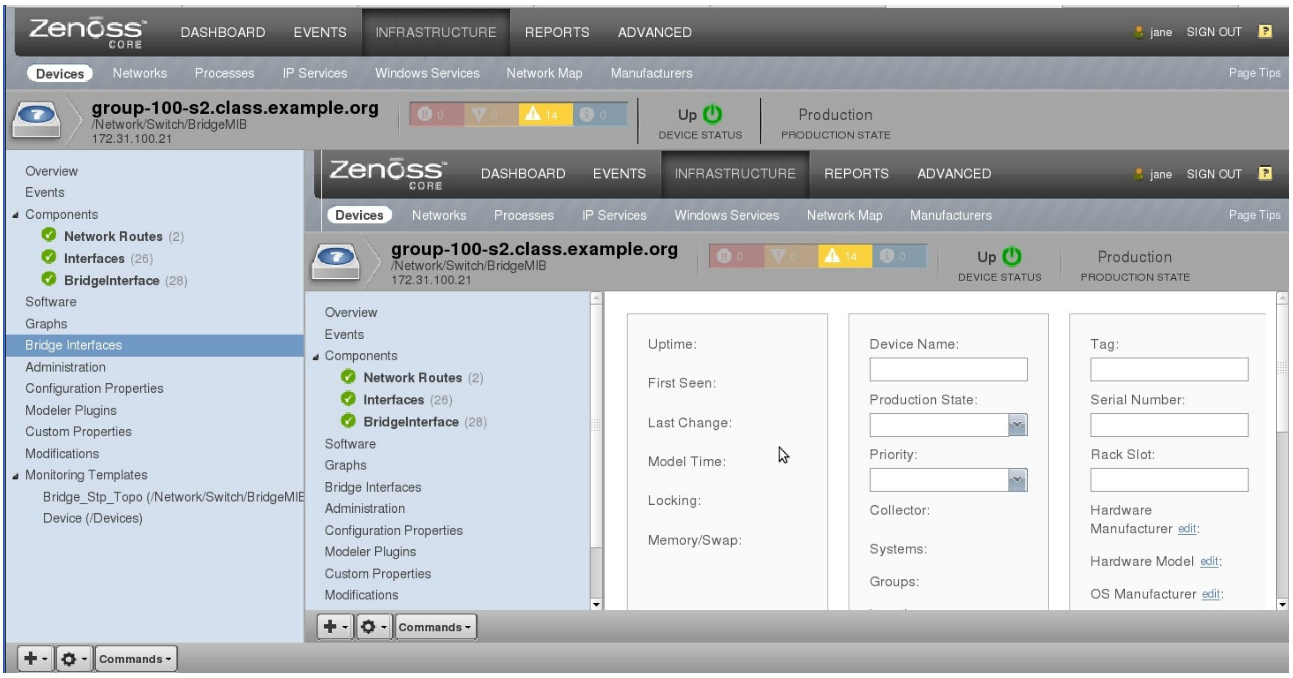
Up — DEVICE STATUS
Production — PRODUCTION STATE

Bridge Interfaces Table

| Port | Remote MAC | Remote Interfaces | Remote Device | Status Value | Port Status |
|---|---|---|---|---|---|
| 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | Learned (3) | 🟢 |
| 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | Learned (3) | 🟢 |
| 3 | 08:4C:BF:C2:79:F5 | | | Learned (3) | 🟢 |
| 4 | 08:4C:92:D0:88:89 | | | Learned (3) | 🟢 |
| 10 | 00:04:C1:02:00:CA | FastEthernet0/10: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 11 | 00:04:C1:02:00:CB | FastEthernet0/11: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 12 | 00:04:C1:02:00:CC | FastEthernet0/12: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 13 | 00:04:C1:02:00:CD | FastEthernet0/13: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 14 | 00:04:C1:02:00:CE | FastEthernet0/14: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 15 | 00:04:C1:02:00:CF | FastEthernet0/15: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 16 | 00:04:C1:02:00:D0 | FastEthernet0/16: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 17 | 00:04:C1:02:00:D1 | FastEthernet0/17: | group-100-s2.class.example.org | Not active (4) | 🔴 |
| 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | Not active (4) | 🔴 |

*Figure 65: Standalone left-hand menu for Bridge Interfaces*

In Zenoss 2, clicking on the relevant *Port* field resulted in the performance graph for that interface (refer back to Figure 54 on page 70). In Zenoss 3, it is unable to follow the factory action to viewBridgeInterface.pt to display the performance graph. It resorts to displaying a default window which is the *Overview* window for the device.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Up — DEVICE STATUS
Production — PRODUCTION STATE

Uptime:
First Seen:
Last Change:
Model Time:
Locking:
Memory/Swap:

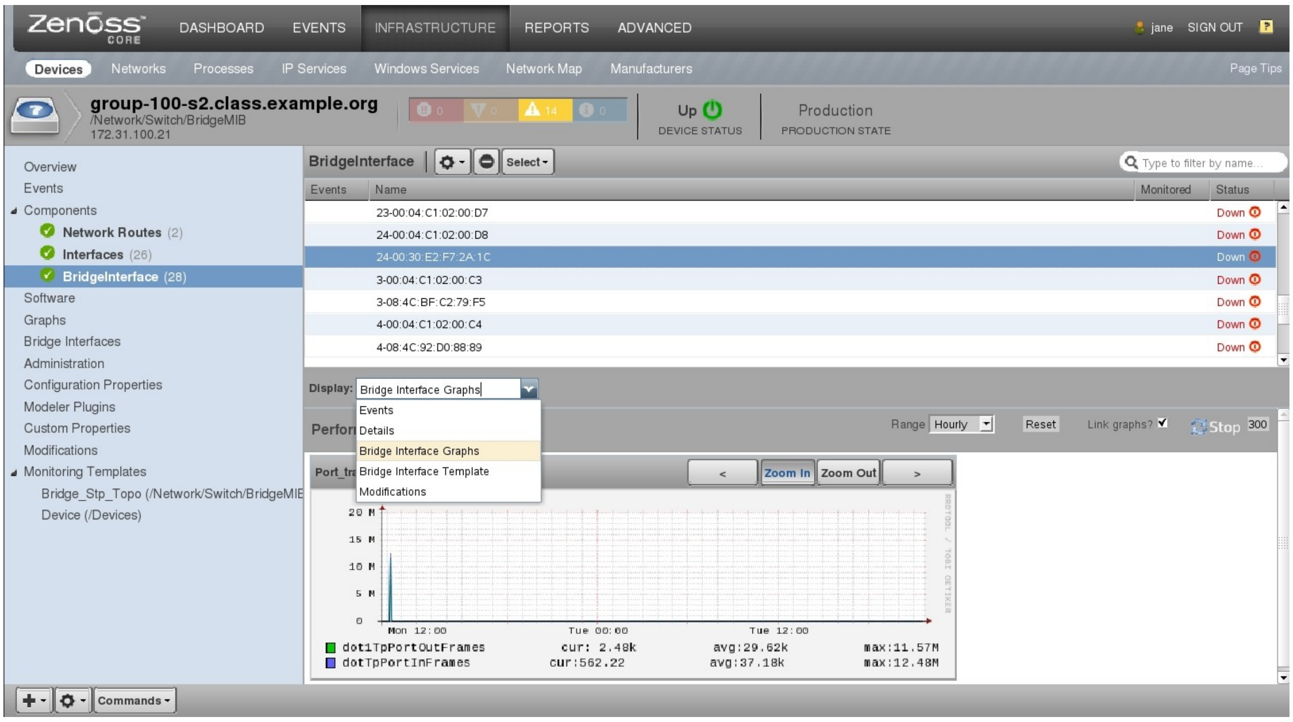
Device Name:
Production State:
Priority:
Collector:
Systems:
Groups:

Tag:
Serial Number:
Rack Slot:
Hardware Manufacturer edit:
Hardware Model edit:
OS Manufacturer edit:

*Figure 66: Result of clicking on a Port link - Overview device window rather than port graph window*

The port performance graphs are not totally lost. The new, redesigned device details page automatically includes a submenu for each component of a device, under the

*Components* menu. Following the *BridgeInterface* submenu from the left-hand *Components* menu produces a panel where the attribute information for the ports is absent (no Remote MAC, Port Status, etc) - a default panel is used for a standard device with Events, Name, Monitored and Status fields; the Status column here does **not** reflect the status of the port; the Name **is** the correct name of the BridgeInterface component.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Up — DEVICE STATUS
Production — PRODUCTION STATE

BridgeInterface

| Events | Name | Monitored | Status |
|---|---|---|---|
| | 23-00:04:C1:02:00:D7 | | Down |
| | 24-00:04:C1:02:00:D8 | | Down |
| | 24-00:30:E2:F7:2A:1C | | Down |
| | 3-00:04:C1:02:00:C3 | | Down |
| | 3-08:4C:BF:C2:79:F5 | | Down |
| | 4-00:04:C1:02:00:C4 | | Down |
| | 4-08:4C:92:D0:88:89 | | Down |

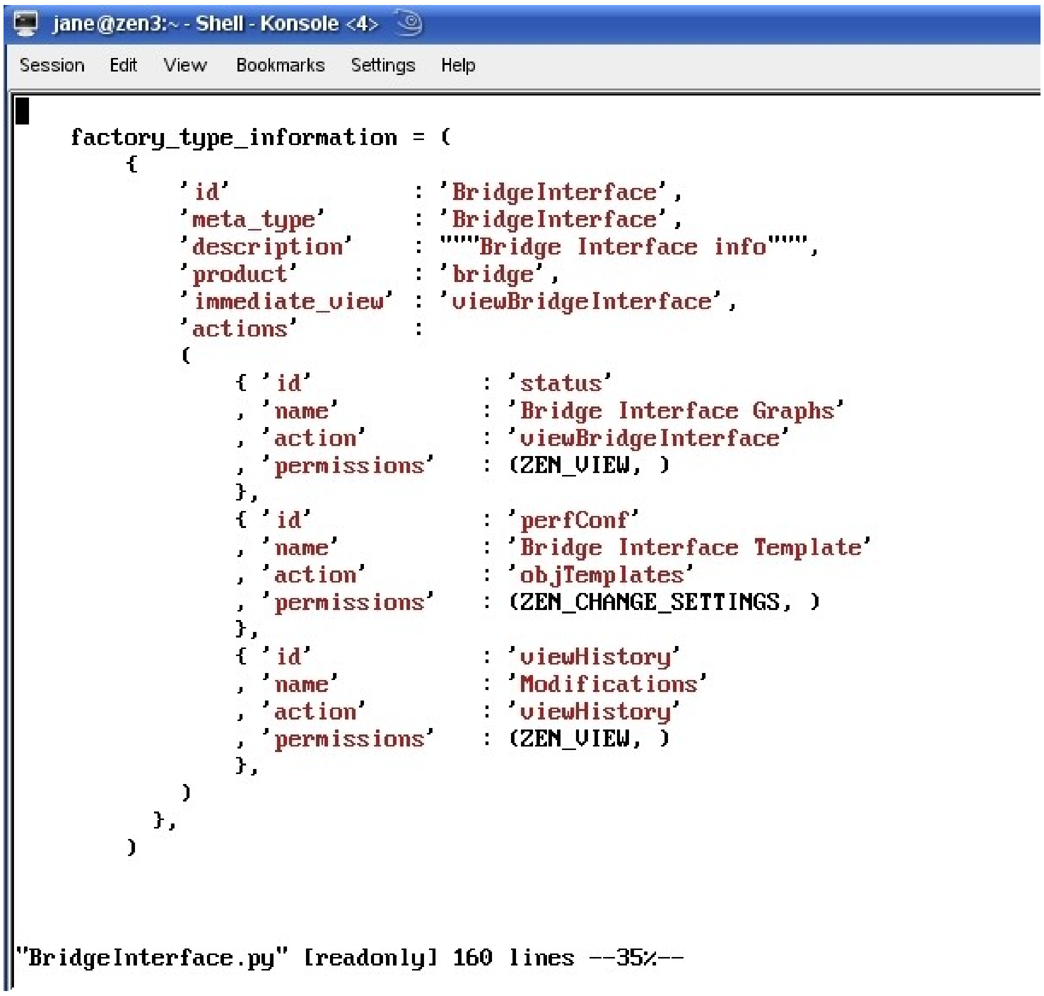
Display: Bridge Interface Graphs

Range: Hourly   Reset   Link graphs? ✓   Stop 300

```
dot1TpPortOutFrames   cur: 2.48k   avg:29.62k   max:11.57M
dotTpPortInFrames     cur:562.22   avg:37.18k   max:12.48M
```

*Figure 67: Default BridgeInterface Component submenu with Display dropdown options*

However the *Display* dropdown offers the factory actions for a BridgeInterface component. Note that the menu options exactly match the name stanza in BridgeInterface.py. In this case, the GUI **does** manage to follow the action stanza for *Bridge Interface Graphs* and find the performance graph defined by viewBridgeInterface.pt. The *Bridge Interface Template* and *Modifications* options also work perfectly, although the options **not** defined by this ZenPack, *Events* and *Details*, do not work.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
    factory_type_information = (
        {
            'id'             : 'BridgeInterface',
            'meta_type'      : 'BridgeInterface',
            'description'    : """Bridge Interface info""",
            'product'        : 'bridge',
            'immediate_view' : 'viewBridgeInterface',
            'actions'        :
            (
                { 'id'            : 'status'
                , 'name'          : 'Bridge Interface Graphs'
                , 'action'        : 'viewBridgeInterface'
                , 'permissions'   : (ZEN_VIEW, )
                },
                { 'id'            : 'perfConf'
                , 'name'          : 'Bridge Interface Template'
                , 'action'        : 'objTemplates'
                , 'permissions'   : (ZEN_CHANGE_SETTINGS, )
                },
                { 'id'            : 'viewHistory'
                , 'name'          : 'Modifications'
                , 'action'        : 'viewHistory'
                , 'permissions'   : (ZEN_VIEW, )
                },
            )
          },
        )


"BridgeInterface.py" [readonly] 160 lines --35%--
```

*Figure 68: Factory information defining menus for a BridgeInterface component*

So, without modifications, the ZenPack can provide the same information in a Zenoss 3 environment; however it is divided between two similar but different menus, each of which has a "working" part and a "non-working" part.

### 4.3.9.3 Improving Bridge Interface information with JavaScript additions

Section 14.1.7 of the Developer's Guide talks about changes necessary to display customised information on the component grid. For the BridgeMIB ZenPack, this means fixing the automatic components submenu so that it shows the port information correctly. Since Zenoss 3 has redesigned all the device details page elements and provided the automatic submenu for device components, it makes sense to adapt to this, rather than trying to get graphs to work on the standalone left-hand *Bridge Interfaces* menu.

In practise, the two skins files become almost redundant and a JavaScript file replaces BridgeDeviceDetail.pt.

Three files are required in the base directory of the ZenPack (...../ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge):

- configure.zcml

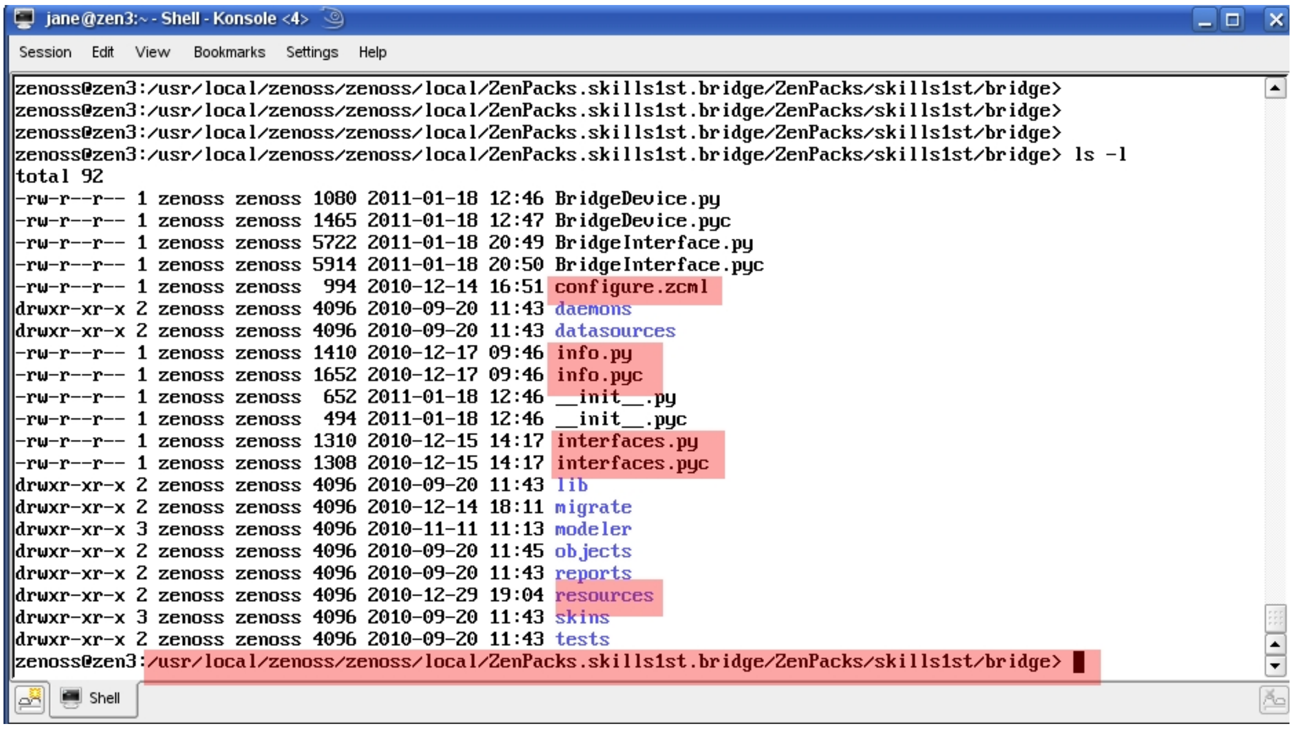
- interfaces.py          (note interfac**es**.py – the guide has this as interface.py)
- info.py

The info file abstracts object attribute information saved in the Zope Object Database (ZODB), that will be displayed to the user. It allows code to be written for display that it is not part of the class definition.

The interfaces file should have basic Zope schema information - formatting details describing the fields of a form for the attributes to populate. Chapter 14 of the Developer's Guide (page 118) suggests that it can be a "barebones", effectively blank file but it is good practise for any object attributes you are exposing in the info object, to be defined in the corresponding interface. interfaces.py controls the fields seen in a component's *Details* dropdown menu. For more information on Zope interfaces, see http://wiki.zope.org/zope3/WhatAreInterfaces .

configure.zcml provides the glue between different components and this exact name will be searched for by the Zope mechanisms. Zope Configuration Markup Language (ZCML) is Zope 3's XML-based component configuration language for "wiring" together application policy and component registrations. It is documented at the Zope site at http://apidoc.zope.org/++apidoc++/ - follow the ZCML link.

The actual detailed code for displaying the Remote Address, Port Status etc for a bridge interface, comes from a JavaScript file (**bridge.js** in this case), which goes in the new **resources** subdirectory of the ZenPack's base directory.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
Shell

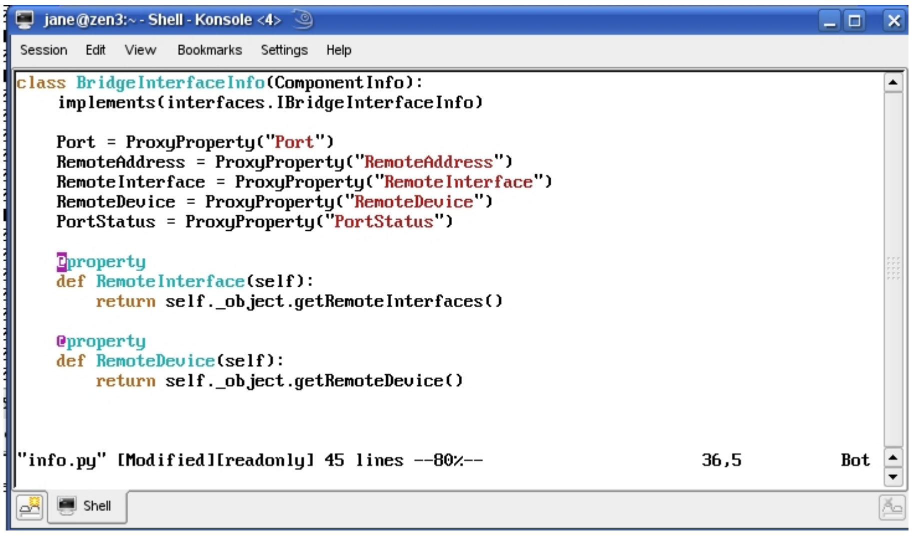
```
zenoss@zen3:/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge>
zenoss@zen3:/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge>
zenoss@zen3:/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge>
zenoss@zen3:/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge> ls -l
total 92
-rw-r--r-- 1 zenoss zenoss 1080 2011-01-18 12:46 BridgeDevice.py
-rw-r--r-- 1 zenoss zenoss 1465 2011-01-18 12:47 BridgeDevice.pyc
-rw-r--r-- 1 zenoss zenoss 5722 2011-01-18 20:49 BridgeInterface.py
-rw-r--r-- 1 zenoss zenoss 5914 2011-01-18 20:50 BridgeInterface.pyc
-rw-r--r-- 1 zenoss zenoss  994 2010-12-14 16:51 configure.zcml
drwxr-xr-x 2 zenoss zenoss 4096 2010-09-20 11:43 daemons
drwxr-xr-x 2 zenoss zenoss 4096 2010-09-20 11:43 datasources
-rw-r--r-- 1 zenoss zenoss 1410 2010-12-17 09:46 info.py
-rw-r--r-- 1 zenoss zenoss 1652 2010-12-17 09:46 info.pyc
-rw-r--r-- 1 zenoss zenoss  652 2011-01-18 12:46 __init__.py
-rw-r--r-- 1 zenoss zenoss  494 2011-01-18 12:46 __init__.pyc
-rw-r--r-- 1 zenoss zenoss 1310 2010-12-15 14:17 interfaces.py
-rw-r--r-- 1 zenoss zenoss 1308 2010-12-15 14:17 interfaces.pyc
drwxr-xr-x 2 zenoss zenoss 4096 2010-09-20 11:43 lib
drwxr-xr-x 2 zenoss zenoss 4096 2010-12-14 18:11 migrate
drwxr-xr-x 3 zenoss zenoss 4096 2010-11-11 11:13 modeler
drwxr-xr-x 2 zenoss zenoss 4096 2010-09-20 11:45 objects
drwxr-xr-x 2 zenoss zenoss 4096 2010-09-20 11:43 reports
drwxr-xr-x 2 zenoss zenoss 4096 2010-12-29 19:04 resources
drwxr-xr-x 3 zenoss zenoss 4096 2010-09-20 11:43 skins
drwxr-xr-x 2 zenoss zenoss 4096 2010-09-20 11:43 tests
zenoss@zen3:/usr/local/zenoss/zenoss/local/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge>
```

*Figure 69: Base directory of ZenPack with new JavaScript-relevant items highlighted*

Note in Figure 69 that the info and interfaces files are Python files so will be compiled on demand into .pyc files, when required.

info.py (and it should be exactly this filename) contains a Python class definition of type ComponentInfo for the **BridgeInterfaceInfo** class. It implements the layout that is found in the **IBridgeInterfaceInfo** class found in interfaces.py.

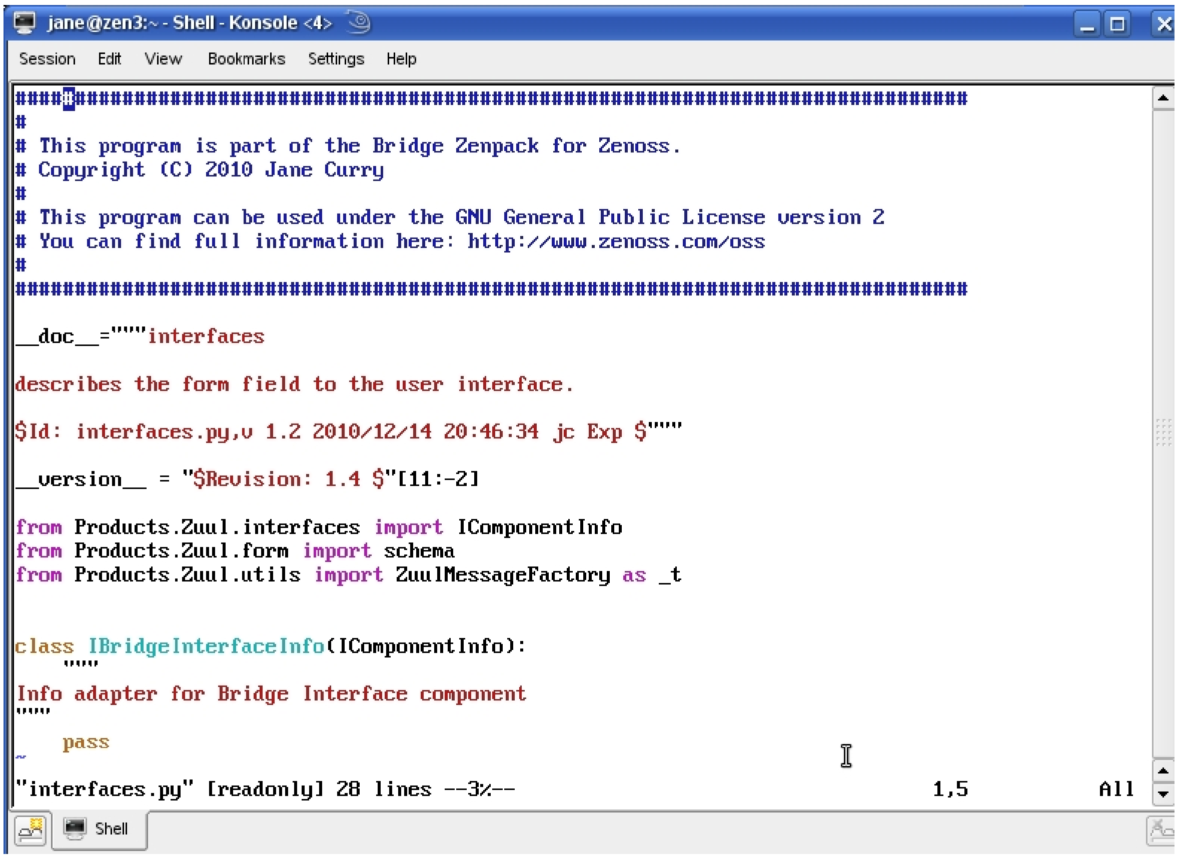
It defines the device attributes that need to be displayed with functions for attributes that need to use methods to retrieve data; these functions (getRemoteInterfaces and getRemoteDevice) are in the object file BridgeInterface.py (as they always have been). The **ProxyProperty** method shuttles data from the Zope Object Database (ZODB) to the info object. There is no formatting in the info.py file.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
Shell

```
class BridgeInterfaceInfo(ComponentInfo):
    implements(interfaces.IBridgeInterfaceInfo)

    Port = ProxyProperty("Port")
    RemoteAddress = ProxyProperty("RemoteAddress")
    RemoteInterface = ProxyProperty("RemoteInterface")
    RemoteDevice = ProxyProperty("RemoteDevice")
    PortStatus = ProxyProperty("PortStatus")

    @property
    def RemoteInterface(self):
        return self._object.getRemoteInterfaces()

    @property
    def RemoteDevice(self):
        return self._object.getRemoteDevice()


"info.py" [Modified][readonly] 45 lines --80%--                36,5          Bot
```

*Figure 70: info.py*

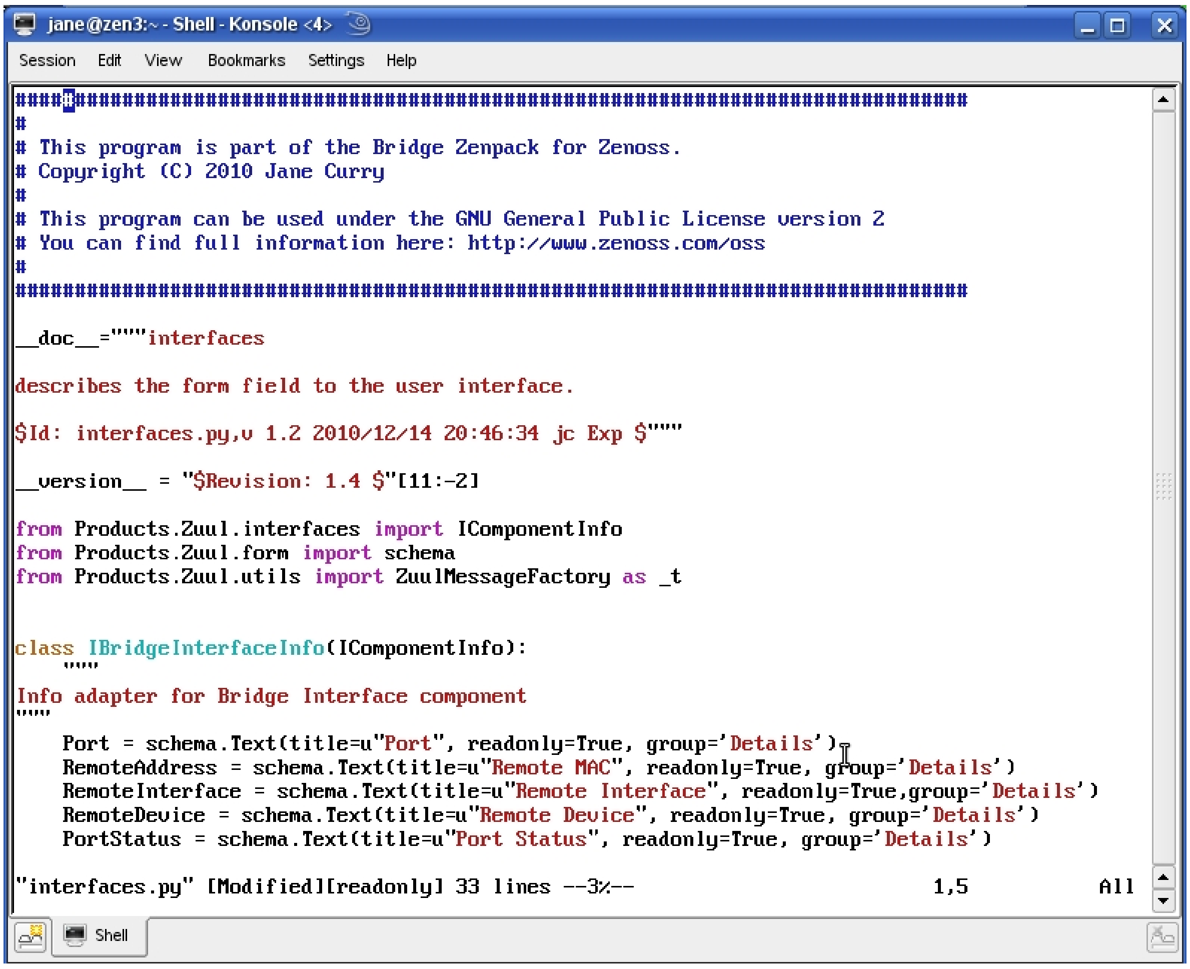
interfaces.py (again, this exact filename) defines the class **IBridgeInterfaceInfo** of type IComponentInfo. Chapter 14 of the Developer's Guide suggests that this file can be a “barebones” interface that does nothing as shown in Figure 71 but this is not good practise.


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
Shell

```
################################################################################
#
# This program is part of the Bridge Zenpack for Zenoss.
# Copyright (C) 2010 Jane Curry
#
# This program can be used under the GNU General Public License version 2
# You can find full information here: http://www.zenoss.com/oss
#
################################################################################

__doc__="""interfaces

describes the form field to the user interface.

$Id: interfaces.py,v 1.2 2010/12/14 20:46:34 jc Exp $"""

__version__ = "$Revision: 1.4 $"[11:-2]

from Products.Zuul.interfaces import IComponentInfo
from Products.Zuul.form import schema
from Products.Zuul.utils import ZuulMessageFactory as _t


class IBridgeInterfaceInfo(IComponentInfo):
    """
Info adapter for Bridge Interface component
"""
    pass
~
"interfaces.py" [readonly] 28 lines --3%--                        1,5           All
```

*Figure 71: "barebones" interfaces.py – not recommended*

The interfaces file should should provide Zope schema information to help generate the fields of a form, with appropriate types such as int, string, etc.  The fields defined in interfaces.py are used and populated in the *Details* dropdown menu from *Display*. If a "barebones" interfaces.py is used then the only field in the *Details* dropdown will be Status (which is nothing to do with Port Status).

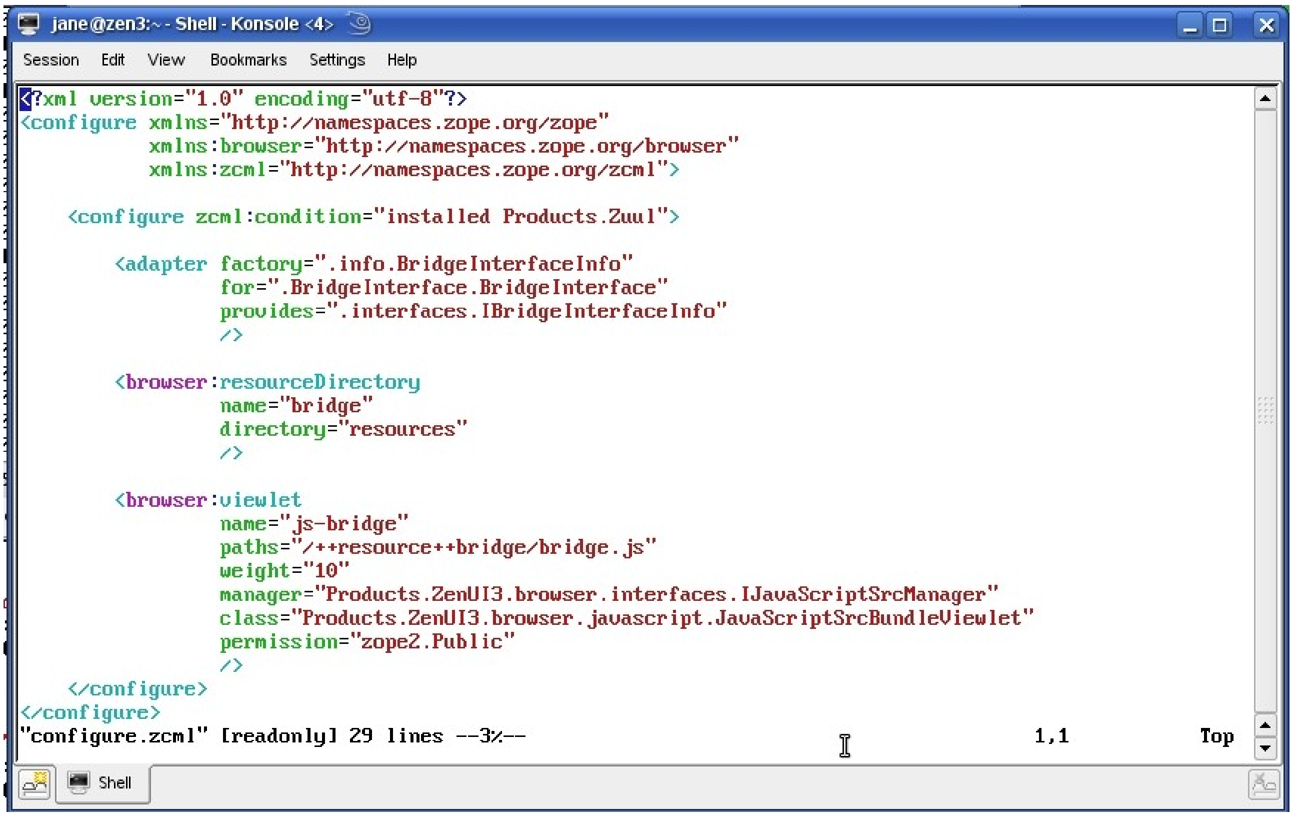

jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
Shell

```
#################################################################################
#
# This program is part of the Bridge Zenpack for Zenoss.
# Copyright (C) 2010 Jane Curry
#
# This program can be used under the GNU General Public License version 2
# You can find full information here: http://www.zenoss.com/oss
#
#################################################################################

__doc__="""interfaces

describes the form field to the user interface.

$Id: interfaces.py,v 1.2 2010/12/14 20:46:34 jc Exp $"""

__version__ = "$Revision: 1.4 $"[11:-2]

from Products.Zuul.interfaces import IComponentInfo
from Products.Zuul.form import schema
from Products.Zuul.utils import ZuulMessageFactory as _t


class IBridgeInterfaceInfo(IComponentInfo):
    """
Info adapter for Bridge Interface component
"""
    Port = schema.Text(title=u"Port", readonly=True, group='Details')
    RemoteAddress = schema.Text(title=u"Remote MAC", readonly=True, group='Details')
    RemoteInterface = schema.Text(title=u"Remote Interface", readonly=True,group='Details')
    RemoteDevice = schema.Text(title=u"Remote Device", readonly=True, group='Details')
    PortStatus = schema.Text(title=u"Port Status", readonly=True, group='Details')

"interfaces.py" [Modified][readonly] 33 lines --3%--                    1,5          All
```

*Figure 72: interfaces.py following "good practise" with schema information*

The **adapter** stanza in configure.zcml links the info file and interfaces file with the device component class. In Figure 73:

- The **factory** field must match the class defined in info.py
- The **provides** field must match the class defined in interfaces.py
- The **for** field must match the device component class, BridgeInterface, defined in the file BridgeInterface.py – hence BridgeInterface.BridgeInterface .


jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help
Shell

```
<?xml version="1.0" encoding="utf-8"?>
<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:browser="http://namespaces.zope.org/browser"
           xmlns:zcml="http://namespaces.zope.org/zcml">

    <configure zcml:condition="installed Products.Zuul">

        <adapter factory=".info.BridgeInterfaceInfo"
                 for=".BridgeInterface.BridgeInterface"
                 provides=".interfaces.IBridgeInterfaceInfo"
                 />

        <browser:resourceDirectory
                 name="bridge"
                 directory="resources"
                 />

        <browser:viewlet
                 name="js-bridge"
                 paths="/++resource++bridge/bridge.js"
                 weight="10"
                 manager="Products.ZenUI3.browser.interfaces.IJavaScriptSrcManager"
                 class="Products.ZenUI3.browser.javascript.JavaScriptSrcBundleViewlet"
                 permission="zope2.Public"
                 />
    </configure>
</configure>
"configure.zcml" [readonly] 29 lines --3%--                     1,1          Top
```

*Figure 73: configure.zcml*

The **browser:resourceDirectory** stanza indicates where to find JavaScript files.

- The **name** (namespace) field must match the last part of the ZenPack name – *bridge*
- The **directory** field is the subdirectory from the base directory of the ZenPack - *resources*

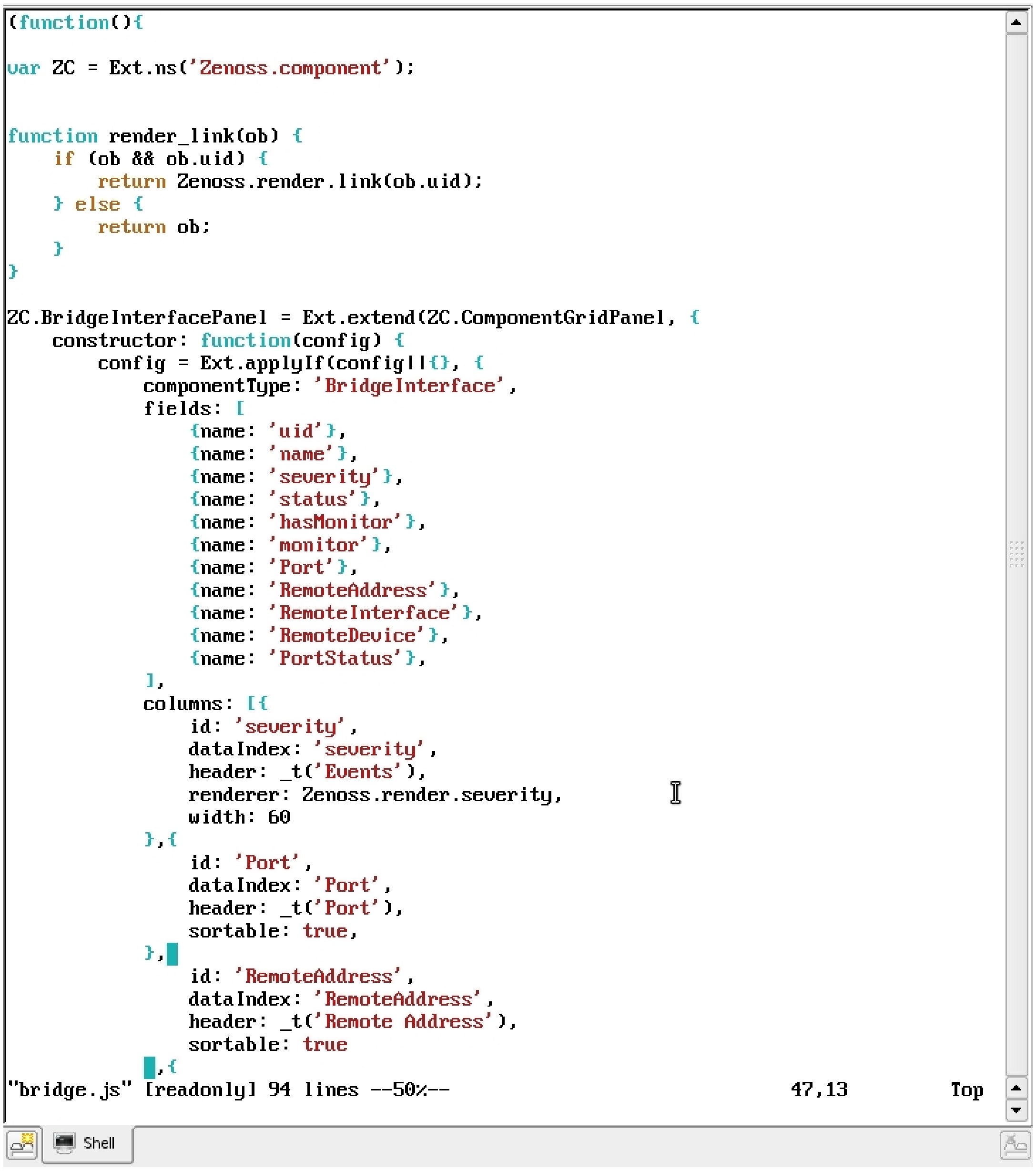
The **browser:viewlet** stanza defines:

- The **name** (namespace) for this viewlet – *js-bridge* – where *bridge* (in this case) must match the last part of the ZenPack name
- The **paths** field indicates the JavaScript file(s) that define the page layout – *bridge.js* in the *bridge* ZenPack's resource directory ie. ...../ *ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/resources*
- The **class** field should be *Products.ZenUI3.browser.javascript.JavaScriptSrcBundleViewlet* if the paths field has one or more files listed. It must be this where the paths field has **multiple** files, space-separated; it could be *Products.ZenUI3.browser.javascript.JavaScriptSrcViewlet* for a single path file.
- The **weight** field indicates the order of multiple viewlets where 1 would be at the top. Although this field seems irrelevant with a single viewlet, removing the line causes the *Bridge Interfaces* Component submenu to fail (reverting to

the default device view – the attributes defined by BaseComponentColModel – see later).

- The **permission** field is mandatory

The JavaScript file, *bridge.js* in the *resources* subdirectory, bears close comparison with the old *BridgeDeviceDetail.pt* page template skins file shown in Figure 57, Figure 58 and Figure 59, starting on page 72.



```
(function(){

var ZC = Ext.ns('Zenoss.component');


function render_link(ob) {
    if (ob && ob.uid) {
        return Zenoss.render.link(ob.uid);
    } else {
        return ob;
    }
}

ZC.BridgeInterfacePanel = Ext.extend(ZC.ComponentGridPanel, {
    constructor: function(config) {
        config = Ext.applyIf(config||{}, {
            componentType: 'BridgeInterface',
            fields: [
                {name: 'uid'},
                {name: 'name'},
                {name: 'severity'},
                {name: 'status'},
                {name: 'hasMonitor'},
                {name: 'monitor'},
                {name: 'Port'},
                {name: 'RemoteAddress'},
                {name: 'RemoteInterface'},
                {name: 'RemoteDevice'},
                {name: 'PortStatus'},
            ],
            columns: [{
                id: 'severity',
                dataIndex: 'severity',
                header: _t('Events'),
                renderer: Zenoss.render.severity,
                width: 60
            },{
                id: 'Port',
                dataIndex: 'Port',
                header: _t('Port'),
                sortable: true,
            },{
                id: 'RemoteAddress',
                dataIndex: 'RemoteAddress',
                header: _t('Remote Address'),
                sortable: true
            },{
"bridge.js" [readonly] 94 lines --50%--                    47,13          Top
```

Figure 74: bridge.js JavaScript file defining component table layout (part 1)

The first main line

```
var ZC = Ext.ns('Zenoss.component');
```

ensures that the ZenPack can associate with standard Zenoss components.

The lines:

```
    ZC.BridgeInterfacePanel = Ext.extend(ZC.ComponentGridPanel, {
        constructor: function(config) {
            config = Ext.applyIf(config||{}, {
                componentType: 'BridgeInterface',
```

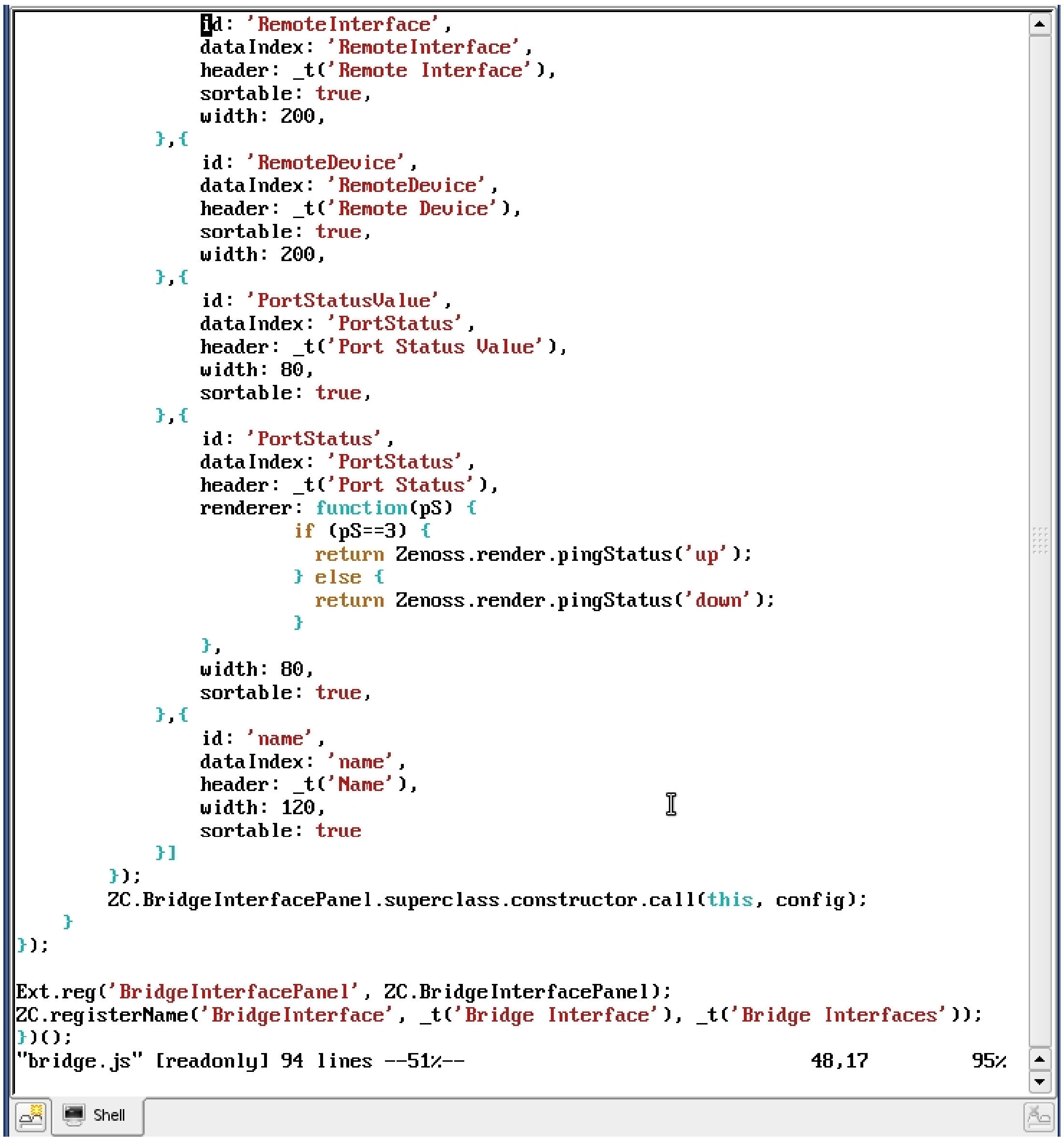
indicate that this is an extension to the standard **ComponentGridPanel** to display the component of type BridgeInterface – the name must be <component name>Panel.

The "fields" lines indicate the object attributes that are either displayed or are needed to make decisions as to what to display – more of this later.

The "columns" stanzas then define column headers and layout for each piece of data to be included in the ComponentGridPanel and can be directly compared with BridgeDeviceDetail.pt.

```
                },{
                    id: 'RemoteInterface',
                    dataIndex: 'RemoteInterface',
                    header: _t('Remote Interface'),
                    sortable: true,
                    width: 200,
                },{
```

The **id** field is a unique identifier and the **dataIndex** field should match the object attribute to be displayed. The column header will be the quoted text in the **header** field. To be able to sort based on a column then **sortable** must be *true*, and a **width** can be supplied.



```
                    id: 'RemoteInterface',
                    dataIndex: 'RemoteInterface',
                    header: _t('Remote Interface'),
                    sortable: true,
                    width: 200,
                },{
                    id: 'RemoteDevice',
                    dataIndex: 'RemoteDevice',
                    header: _t('Remote Device'),
                    sortable: true,
                    width: 200,
                },{
                    id: 'PortStatusValue',
                    dataIndex: 'PortStatus',
                    header: _t('Port Status Value'),
                    width: 80,
                    sortable: true,
                },{
                    id: 'PortStatus',
                    dataIndex: 'PortStatus',
                    header: _t('Port Status'),
                    renderer: function(pS) {
                            if (pS==3) {
                              return Zenoss.render.pingStatus('up');
                            } else {
                              return Zenoss.render.pingStatus('down');
                            }
                    },
                    width: 80,
                    sortable: true,
                },{
                    id: 'name',
                    dataIndex: 'name',
                    header: _t('Name'),
                    width: 120,
                    sortable: true
                }]
            });
            ZC.BridgeInterfacePanel.superclass.constructor.call(this, config);
        }
});

Ext.reg('BridgeInterfacePanel', ZC.BridgeInterfacePanel);
ZC.registerName('BridgeInterface', _t('Bridge Interface'), _t('Bridge Interfaces'));
})();
"bridge.js" [readonly] 94 lines --51%--                        48,17          95%
```

*Figure 75: bridge.js JavaScript file defining component table layout (part 2)*

The Zenoss 2 BridgeDeviceDetail.pt included two columns for port status; one was the numeric value and its textual meaning; the second was a red/green status indicator. The bottom half of bridge.js shows a similar section presenting the numeric value of port status under the header *Port Status Value* and then a red / green indicator, headed *Port Status*. It demonstrates using a different **renderer** to display data. Note also the severity renderer used for the Events column. Basically they use defined icons to represent data rather than the standard text renderer.

```
                },{
                    id: 'PortStatusValue',
                    dataIndex: 'PortStatus',
                    header: _t('Port Status Value'),
                    width: 80,
                    sortable: true,
                },{
                    id: 'PortStatus',
                    dataIndex: 'PortStatus',
                    header: _t('Port Status'),
                    renderer: function(pS) {
                            if (pS==3) {
                              return Zenoss.render.pingStatus('up');
                            } else {
                              return Zenoss.render.pingStatus('down');
                            }
                    },
                    width: 80,
                    sortable: true,
                },{
```

The local function *pS* tests the value of the dataIndex attribute and returns output dependent on the value (remember that a port status of 3 equates to a "Learned" value which the ZenPack defines as "active"). Other standard Zenoss renderers can be found in *$ZENHOME/Products/ZenUI3/browser/resources/js/zenoss/Renderers.js*.

The last 2 lines

```
    Ext.reg('BridgeInterfacePanel', ZC.BridgeInterfacePanel);
    ZC.registerName('BridgeInterface', _t('Bridge Interface'), _t('Bridge Interfaces'));
```

ensure that this extension to the ComponentGridPanel is registered, with the last line associating the component BridgeInterface object with this panel and the Component submenu being entitled *Bridge Interface* if there is only one instance of the component type, or *Bridge Interface**s*** if there are several instances.

Note the "fields" section in Figure 74. There are far more fields there than will actually be displayed. This is because other code elements in the ComponentPanel code test for these values and if they are not defined in the "fields" section then the component grid will revert to the standard (fairly useless) device format. Fields that are required include:

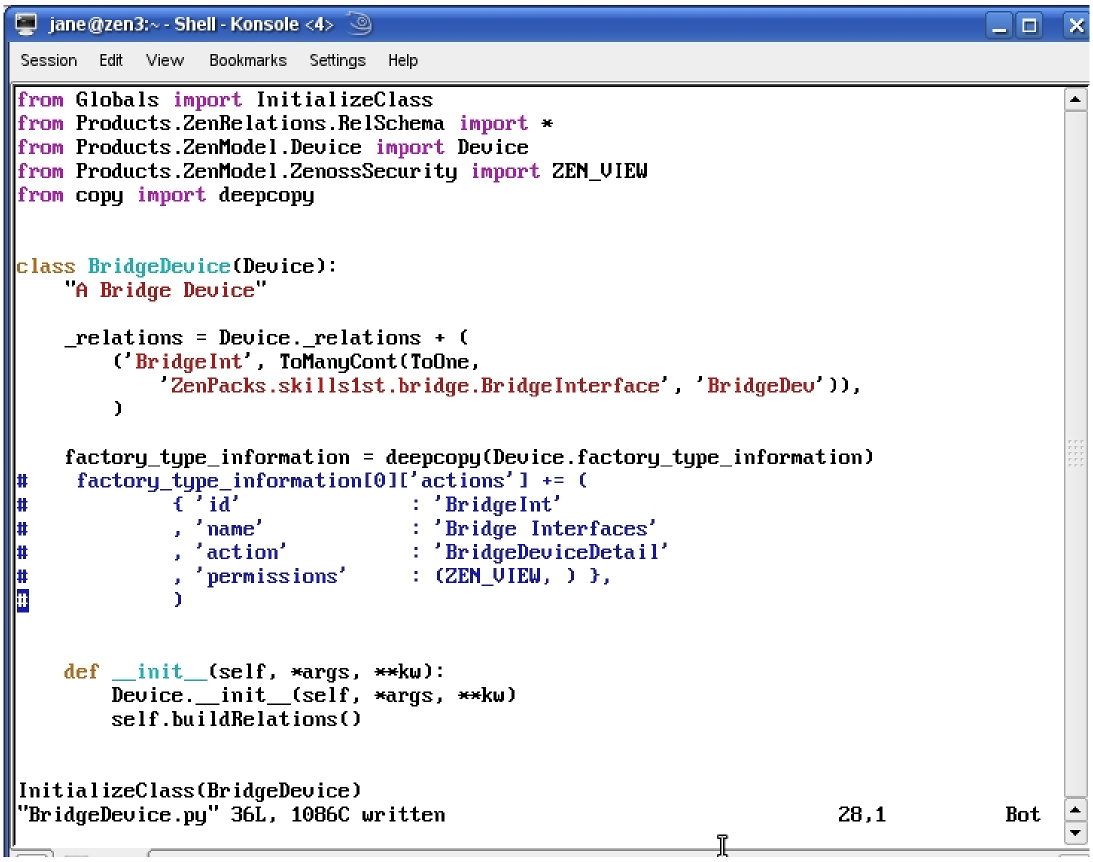
- uid
- name
- monitor
- severity
- status
- hasMonitor

Having added info.py, interfaces.py, configure.zcml to the base ZenPack directory, and bridge.js to the resources subdirectory, simply recycle zopectl to see the effect

```
    zopectl restart
```

The *Bridge Interfaces* submenu of *Components* should now have the correct port information in the top panel, and a Display dropdown with menus to show port graphs, the template for those graphs and Modifications. In fact, it also has a separate, default *Graphs* menu which has the same effect as the customised *Bridge Interface Graphs* menu – more of this later.

To complete the tidying of the Zenoss 3 GUI, the standalone *Bridge Interfaces* menu could be removed by simply commenting out the lines that define the Bridge Interfaces menu, in BridgeDevice.py.

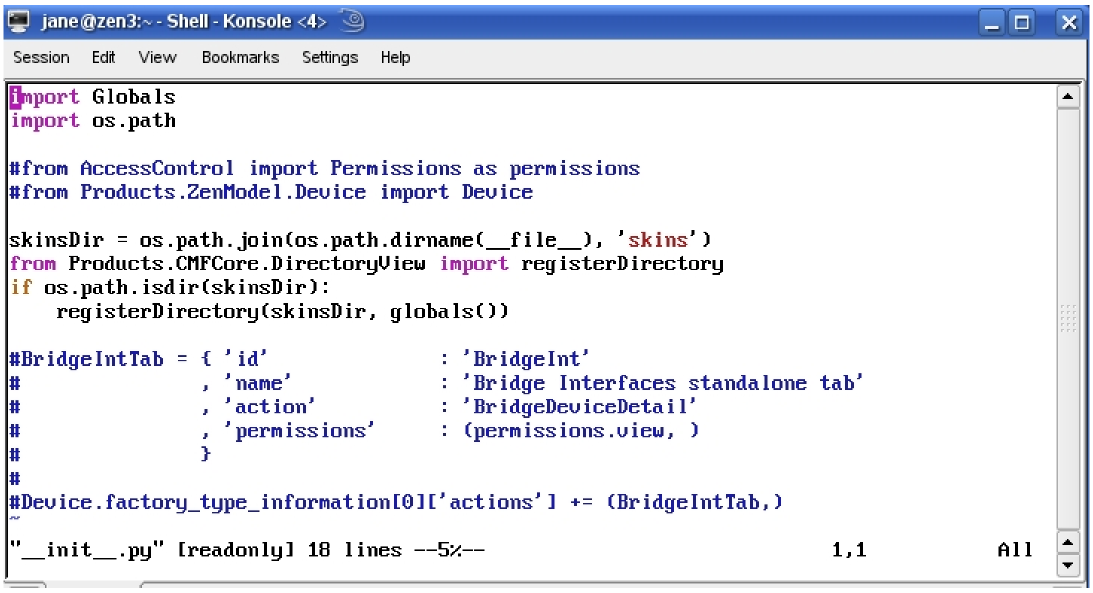

jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
from Globals import InitializeClass
from Products.ZenRelations.RelSchema import *
from Products.ZenModel.Device import Device
from Products.ZenModel.ZenossSecurity import ZEN_VIEW
from copy import deepcopy


class BridgeDevice(Device):
    "A Bridge Device"

    _relations = Device._relations + (
        ('BridgeInt', ToManyCont(ToOne,
            'ZenPacks.skills1st.bridge.BridgeInterface', 'BridgeDev')),
        )

    factory_type_information = deepcopy(Device.factory_type_information)
#    factory_type_information[0]['actions'] += (
#            { 'id'              : 'BridgeInt'
#            , 'name'            : 'Bridge Interfaces'
#            , 'action'          : 'BridgeDeviceDetail'
#            , 'permissions'     : (ZEN_VIEW, ) },
#            )


    def __init__(self, *args, **kw):
        Device.__init__(self, *args, **kw)
        self.buildRelations()


InitializeClass(BridgeDevice)
"BridgeDevice.py" 36L, 1086C written                          28,1          Bot
```

*Figure 76: BridgeDevice.py object file with the Bridge Interfaces menu commented out*

Both zenhub and zopectl should be recycled after changing the object file before refreshing the GUI window.

Incidentally, the Zenoss Developer's Guide Chapter 14 suggests adding standalone left-hand menus for a device, by editing the __init__.py of a ZenPack. The code sample that they offer on page 114 needs a couple of extra imports and, although this mechanism works fine, the new menu will be added for **all** device types, which may not be the desired effect.



jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

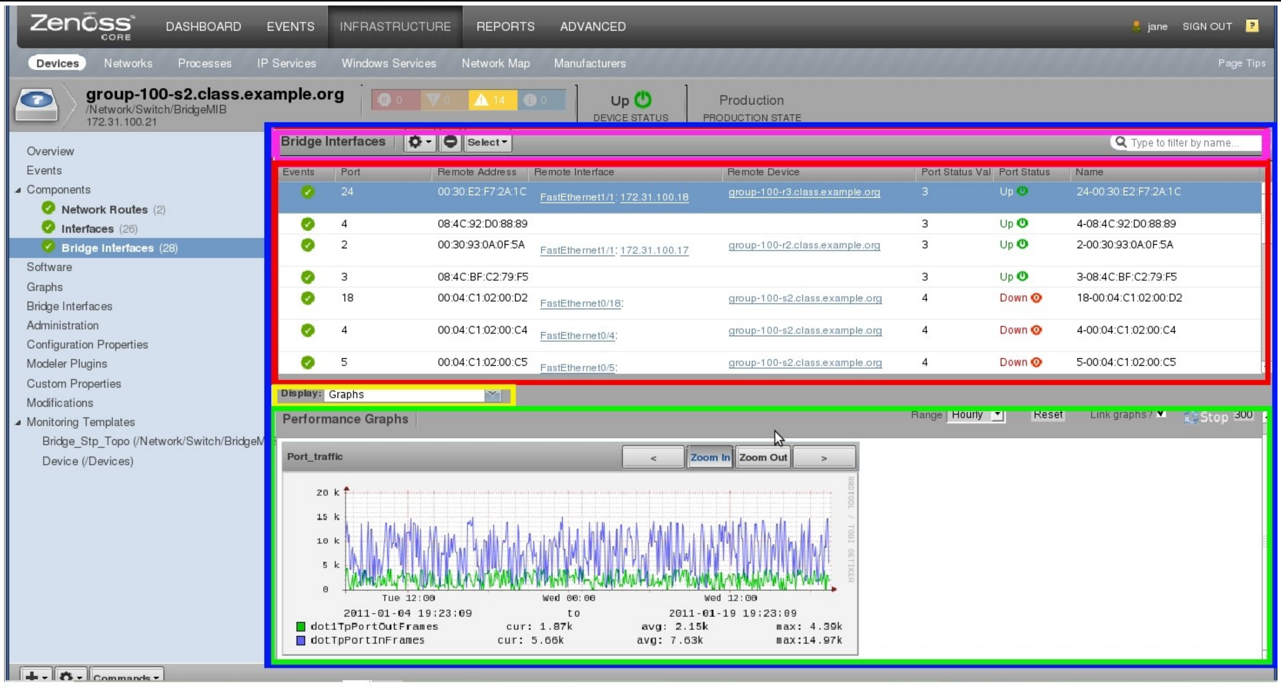
```
import Globals
import os.path

#from AccessControl import Permissions as permissions
#from Products.ZenModel.Device import Device

skinsDir = os.path.join(os.path.dirname(__file__), 'skins')
from Products.CMFCore.DirectoryView import registerDirectory
if os.path.isdir(skinsDir):
    registerDirectory(skinsDir, globals())

#BridgeIntTab = { 'id'               : 'BridgeInt'
#                , 'name'             : 'Bridge Interfaces standalone tab'
#                , 'action'           : 'BridgeDeviceDetail'
#                , 'permissions'      : (permissions.view, )
#                }
#
#Device.factory_type_information[0]['actions'] += (BridgeIntTab,)
~
"__init__.py" [readonly] 18 lines --5%--                        1,1           All
```

*Figure 77: __init__.py for ZenPack with commented out lines for extra left-hand menu*

Note in Figure 77 the commented out lines to add a standalone left-hand menu.  The two extra import statements are required.

### 4.3.9.4 Understanding the Component Panel in Zenoss 3

Many ZenPacks extend device component capabilities so understanding the new component panel in Zenoss 3 is important.  The new code that defines it is in **$ZENHOME/Products/ZenUI3/browser/resources/js/zenoss**.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Bridge Interfaces

| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | Name |
|---|---|---|---|---|---|---|---|
| ✓ | 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | 3 | Up | 24-00:30:E2:F7:2A:1C |
| ✓ | 4 | 08:4C:92:D0:88:89 | | | 3 | Up | 4-08:4C:92:D0:88:89 |
| ✓ | 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | 3 | Up | 2-00:30:93:0A:0F:5A |
| ✓ | 3 | 08:4C:BF:C2:79:F5 | | | 3 | Up | 3-08:4C:BF:C2:79:F5 |
| ✓ | 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | 4 | Down | 18-00:04:C1:02:00:D2 |
| ✓ | 4 | 00:04:C1:02:00:C4 | FastEthernet0/4: | group-100-s2.class.example.org | 4 | Down | 4-00:04:C1:02:00:C4 |
| ✓ | 5 | 00:04:C1:02:00:C5 | FastEthernet0/5: | group-100-s2.class.example.org | 4 | Down | 5-00:04:C1:02:00:C5 |

Display: Graphs

Performance Graphs

Port_traffic

```
2011-01-04 19:23:09          to          2011-01-19 19:23:09
dot1TpPortOutFrames      cur: 1.87k      avg: 2.15k      max: 4.36k
dotTpPortInFrames        cur: 5.66k      avg: 7.63k      max:14.97k
```

| | |
|---|---|
| 🟦 | ZC.ComponentPanel |
| 🟪 | Title Bar |
| 🟥 | ZC.ComponentGridPanel |
| 🟨 | Zenoss.DetailNavCombo |
| 🟩 | Zenoss.ContextCardPanel |

*Figure 78: ComponentPanel diagram*

Examining ComponentPanel.js (line numbers here are given for Zenoss 3.0.3):

- **Zenoss.nav.register** from lines 46-122 sets up the default dropdown menus from the *Display* DetailNavCombo box – *Graphs, Events* and *Details*. This is why the BridgeMIB ZenPack has a *Graphs* menu in addition to its own *Bridge Interface Graphs* menu.
- **ZC.ComponentDetailNav** from lines 124 to 183, is concerned with augmenting the *Display* dropdown menu and explicitly prohibits menu items with the names *status, events, resetcommunity,pushconfig, objtemplates,modeldevice* and *historyevents*.
- **ZC.ComponentPanel** runs from lines 186 to 334. Fundamentally there are four main areas inside the entire **Component Panel** (outlined in blue in Figure 78):
  - The Title Bar (tbar) outlined in pink
  - The Component Grid Panel with attribute values for each instance of a component, outlined in red
  - The title bar of the bottom half of the window is the text *Display* and a DetailNavCombo dropdown box to select the data to be seen at the bottom. This is outlined in yellow. This section prohibits the display of the *Graphs* dropdown menu if the monitor attribute is not set for the object. It also

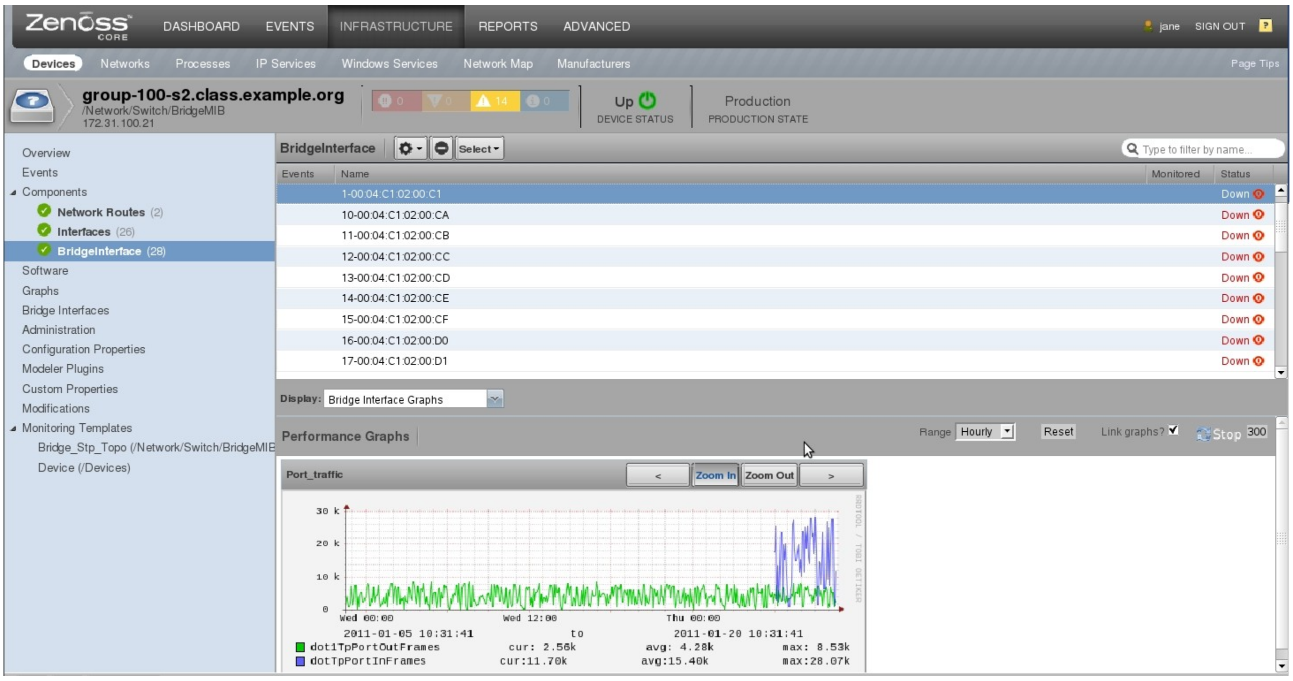
filters out any dropdown menu items that match the list given above under ZC.ComponentDetailNav.

  - The Context Card Panel is the bottom window with graphs, events, details, etc and is outlined in green.

- **ZC.ComponentGridPanel** (lines 337 – 412) defines the container for the top part of the component panel – the Component Grid Panel – and fills it, by default, with a BaseComponent Store that uses the BaseComponentColModel attributes, ensuring alternate striping for each row (stripeRows:  true)
- **ZC.BaseComponentStore** (lines 337 – 412) defines the default object attribute fields that will be used to help construct the top panel, unless they are overridden by custom JavaScript via *config.fields*.  Omitting any of these fields in custom JavaScript may lead to unpredictable results.

```
ZC.BaseComponentStore = Ext.extend(Ext.ux.grid.livegrid.Store, {
    constructor: function(config) {
        var fields = config.fields || [
            {name: 'uid'},
            {name: 'severity'},
            {name: 'name'},
            {name: 'usesMonitorAttribute'},
            {name: 'monitor'},
            {name: 'monitored'},
            {name: 'status'}
        ];
```

- **ZC.BaseComponentColModel** (lines 467 – 500) define the default attribute fields that will be displayed in the top ComponentGridPanel – *Events, Name, Monitored* and *Status*.  This will be the ComponentGridPanel that is used if something in custom JavaScript is not able to be interpreted correctly – see Figure 79.
- The remaining definitions **ZC.IPInterfacePanel, ZC.WinServicePanel, ZC.IpRouteEntryPanel, ZC.IpServicePanel, ZC.OSProcessPanel, ZC.FileSystemPanel** and **ZC.CPUPanel** each define the specific ComponentGridPanel for the individual, standard component objects.  Note that the code here would be good samples from which to start writing custom JavaScript for new component objects.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

0 0 14 0 | Up DEVICE STATUS | Production PRODUCTION STATE

BridgeInterface

| Events | Name | Monitored | Status |
|---|---|---|---|
| | 1-00:04:C1:02:00:C1 | | Down |
| | 10-00:04:C1:02:00:CA | | Down |
| | 11-00:04:C1:02:00:CB | | Down |
| | 12-00:04:C1:02:00:CC | | Down |
| | 13-00:04:C1:02:00:CD | | Down |
| | 14-00:04:C1:02:00:CE | | Down |
| | 15-00:04:C1:02:00:CF | | Down |
| | 16-00:04:C1:02:00:D0 | | Down |
| | 17-00:04:C1:02:00:D1 | | Down |

Display: Bridge Interface Graphs

```
2011-01-05 10:31:41            to            2011-01-20 10:31:41
🟩 dot1TpPortOutFrames     cur: 2.56k     avg: 4.28k     max: 8.53k
🟦 dotTpPortInFrames       cur:11.70k     avg:15.40k     max:28.07k
```

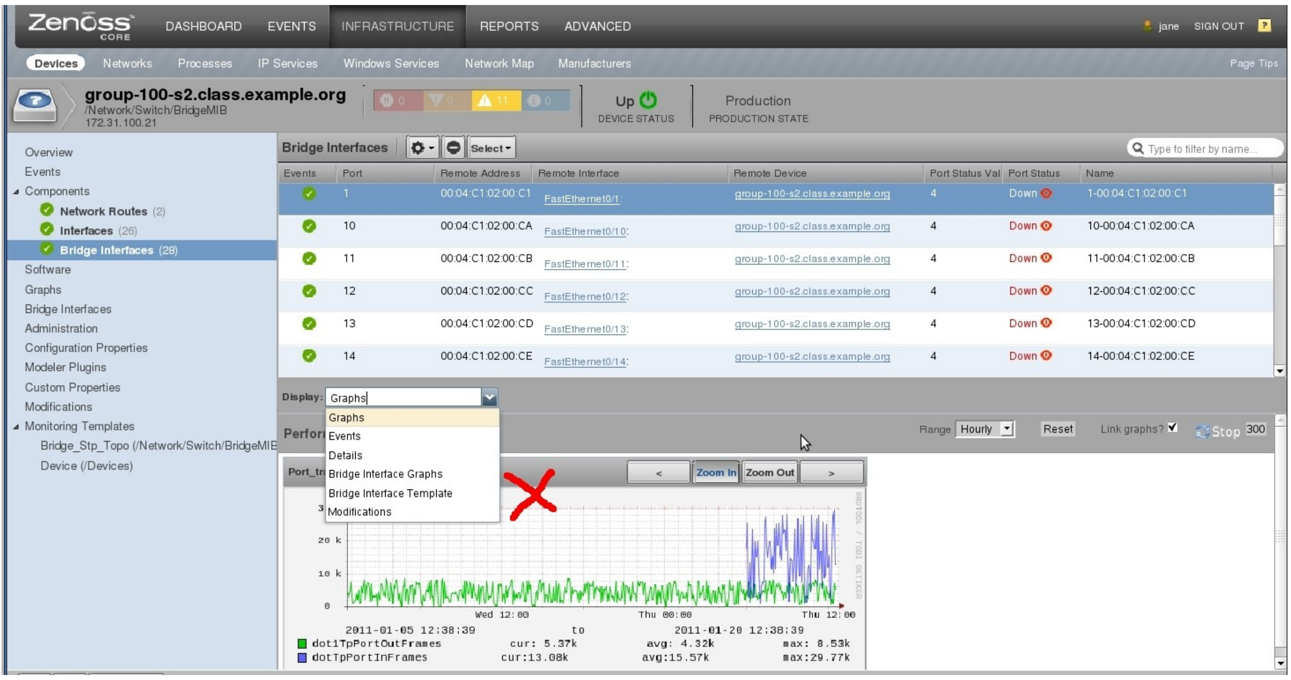
*Figure 79: Default ComponentGridPanel (with BaseComponentColModel) attributes*

When the **ZC.ComponentPanel** constructor executes, the upper space of the window (designated by the **red** border) is left empty. A **ZC.ComponentGridPanel** is loaded into this space by the **ZC.ComponentPanel.*setContext*** method. The code begins on line 292 of ComponentPanel.js. One of the parameters passed to this method is ***type***, which is a string containing the component object class name. The code searches for a registered component with the name ***type*** + "Panel". So if your component is named *BridgeInterface*, the code searches for a registered object named *BridgeInterfacePanel* and uses the code to fill in this upper slot. Note that all the definitions for standard components at the end of ComponentPanel.js follow this model – IpInterfacePanel, FileSystemPanel, etc.

The lower panel is a **Zenoss.ContextCardPanel** (designated by the **green** border). The code for this class begins on line 65 of ContextCardButtonPanel.js (in the same directory as ComponentPanel.js). The configuration of this object begins on line 213 of ComponentPanel.js. Notice that the **tbar** or top toolbar is defined beginning on line 220. The second item in this toolbar is the **Zenoss.DetailNavCombo** (designated by the **yellow** border). This is the drop-down combo box that lists the different options available to be displayed in the **Zenoss.ContextCardPanel**. It is defined beginning on line 580 of SubselectionPanel.js.

Now it gets interesting. The **Zenoss.DetailNavCombo.*setContext*** method calls **Zenoss.remote.DetailNavRouter.*getDetailNavConfigs***. **Zenoss.remote.DetailNavRouter** is a python object. Its class definition is in $ZENHOME/Products/Zuul/routers/nav.py. It appears that this python object is accessible to the JavaScript ExtJS library code via ZCML "wiring" beginning on line 61 of the $ZENHOME/Products/Zuul/routers/configure.zcml file. I do not fully understand how the method call to ***getDetailNavConfigs*** by the ***setContext*** method works because the arguments don't seem to match. Nevertheless, the ***getDetailNavConfigs*** method accesses the component object and calls the ***zentinelTabs*** method on the object. This, returns the list of dictionaries found in **component.factory_type_information['actions']**. These actions are converted to menu items.

In other words, menu items defined in the factory definitions of objects, and their associated skins files, from old-style Zenoss 2, are incorporated into the *Display* dropdown menu in the Component Panel in Zenoss 3.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Up DEVICE STATUS

Production PRODUCTION STATE

Bridge Interfaces

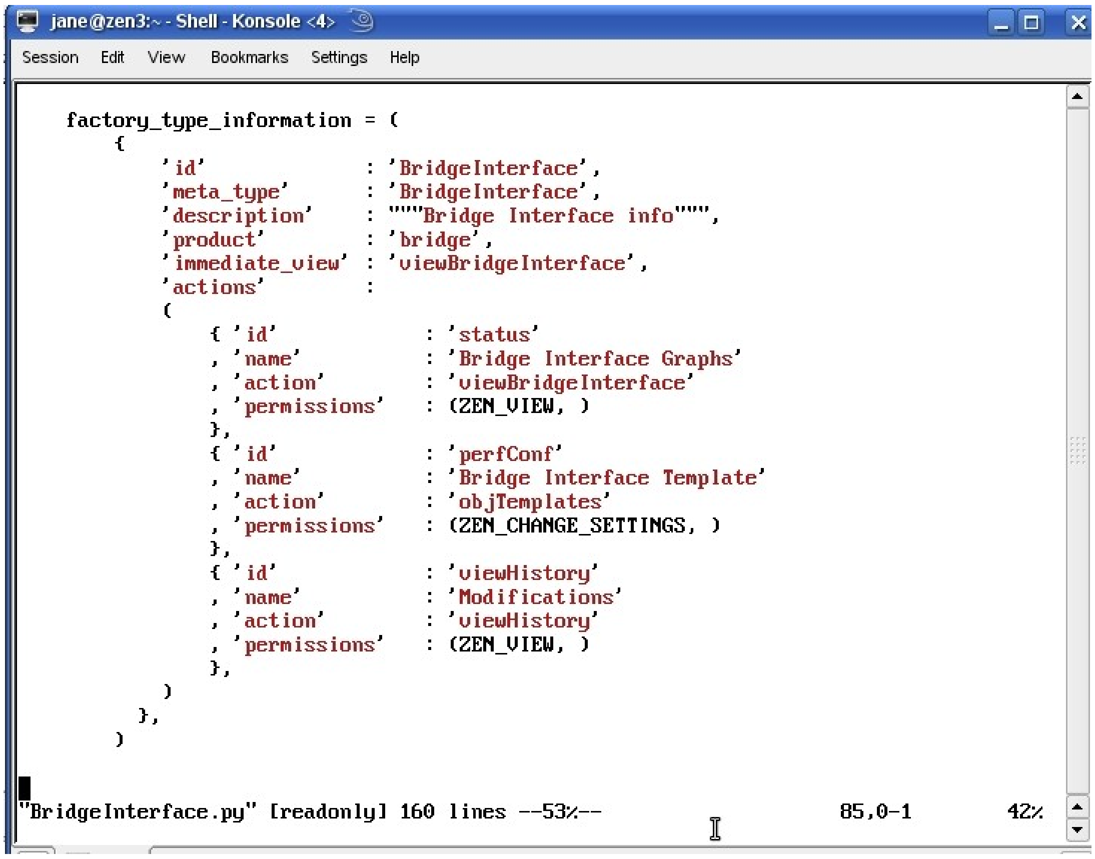
| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | Name |
|---|---|---|---|---|---|---|---|
| ✓ | 1 | 00:04:C1:02:00:C1 | FastEthernet0/1: | group-100-s2.class.example.org | 4 | Down | 1-00:04:C1:02:00:C1 |
| ✓ | 10 | 00:04:C1:02:00:CA | FastEthernet0/10: | group-100-s2.class.example.org | 4 | Down | 10-00:04:C1:02:00:CA |
| ✓ | 11 | 00:04:C1:02:00:CB | FastEthernet0/11: | group-100-s2.class.example.org | 4 | Down | 11-00:04:C1:02:00:CB |
| ✓ | 12 | 00:04:C1:02:00:CC | FastEthernet0/12: | group-100-s2.class.example.org | 4 | Down | 12-00:04:C1:02:00:CC |
| ✓ | 13 | 00:04:C1:02:00:CD | FastEthernet0/13: | group-100-s2.class.example.org | 4 | Down | 13-00:04:C1:02:00:CD |
| ✓ | 14 | 00:04:C1:02:00:CE | FastEthernet0/14: | group-100-s2.class.example.org | 4 | Down | 14-00:04:C1:02:00:CE |

Display: Graphs

```
2011-01-05 12:38:39          to          2011-01-20 12:38:39
dot1TpPortOutFrames      cur: 5.37k      avg: 4.32k      max: 8.53k
dotTpPortInFrames        cur:13.08k      avg:15.57k      max:29.77k
```

*Figure 80: Display dropdown DetailNavCombo for BridgeInterface with 3 m3nus inherited from Zenoss 2 definitions*

This is why the BridgeInterface component panel dropdown includes *Bridge Interface Graphs, Bridge Interface Template* and *Modifications*.

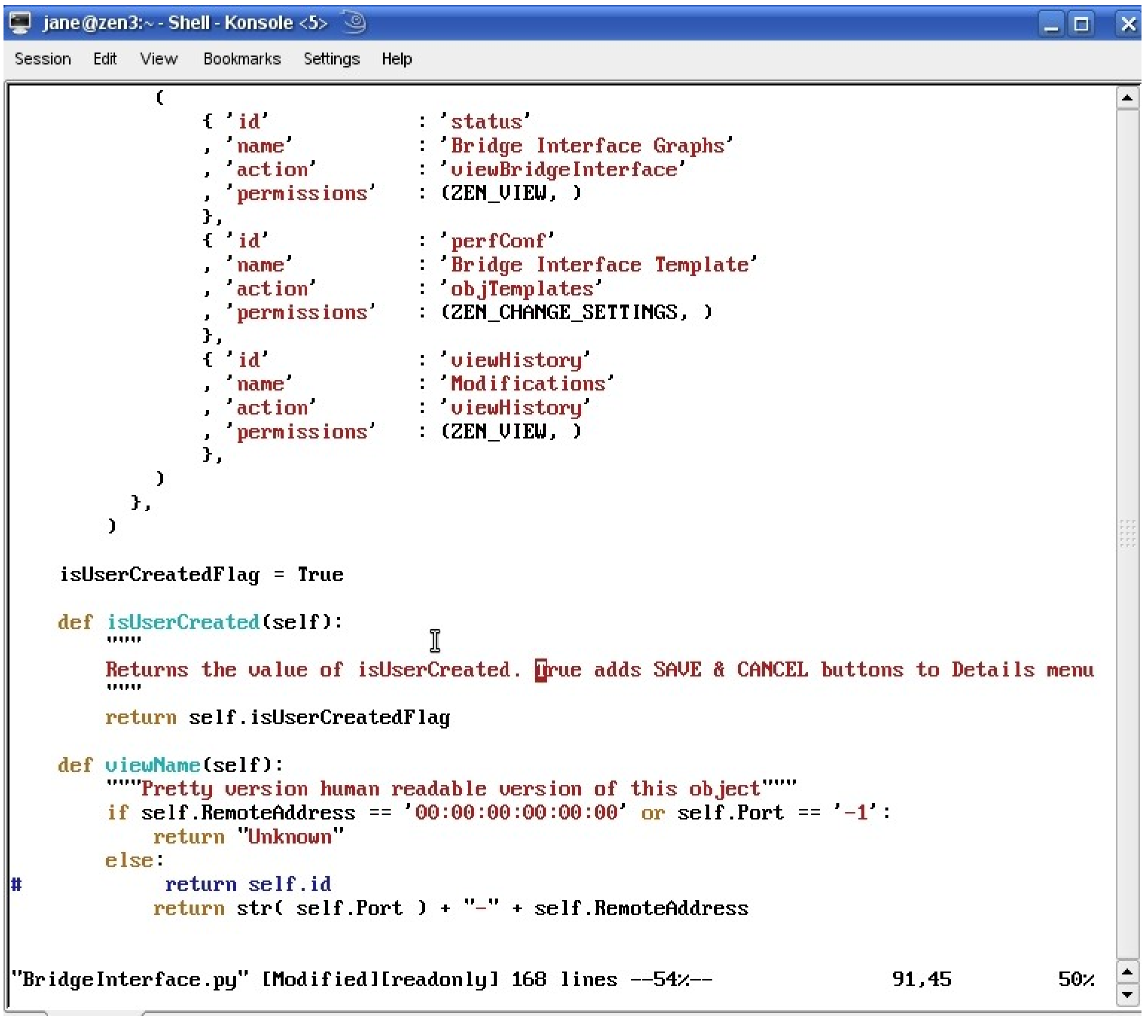

jane@zen3:~ - Shell - Konsole <4>
Session Edit View Bookmarks Settings Help

```
    factory_type_information = (
        {
            'id'             : 'BridgeInterface',
            'meta_type'      : 'BridgeInterface',
            'description'    : """Bridge Interface info""",
            'product'        : 'bridge',
            'immediate_view' : 'viewBridgeInterface',
            'actions'        :
            (
                { 'id'              : 'status'
                , 'name'            : 'Bridge Interface Graphs'
                , 'action'          : 'viewBridgeInterface'
                , 'permissions'     : (ZEN_VIEW, )
                },
                { 'id'              : 'perfConf'
                , 'name'            : 'Bridge Interface Template'
                , 'action'          : 'objTemplates'
                , 'permissions'     : (ZEN_CHANGE_SETTINGS, )
                },
                { 'id'              : 'viewHistory'
                , 'name'            : 'Modifications'
                , 'action'          : 'viewHistory'
                , 'permissions'     : (ZEN_VIEW, )
                },
            )
          },
        )

"BridgeInterface.py" [readonly] 160 lines --53%--                85,0-1        42%
```

*Figure 81: factory_type_information menu definitions for BridgeInterface object*

At this point, it is unclear how to selectively affect the options in the DetailNavCombo dropdown without linking to the old factory definitions and skins files.

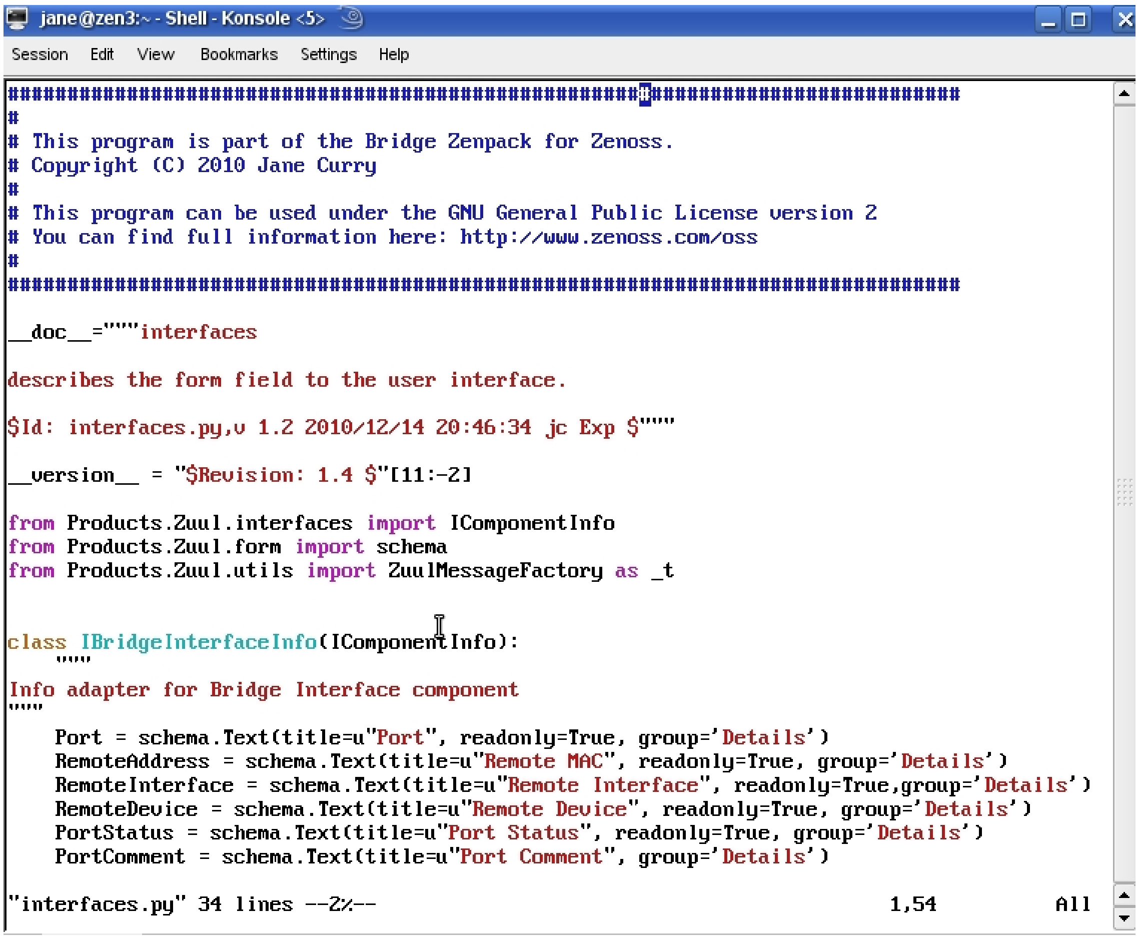
The default *Details* window from the *Display* dropdown is controlled by the fields in *interfaces.py*. Typically they are read-only fields but they could be made read-write by ensuring that the component object class file contains the *isUserCreated* method that returns *True* – see the top of $ZENHOME/Products/ZenModel/OSComponent.py for an example. Adding the lines shown in Figure 82 adds *SAVE* and *CANCEL* buttons to the Details dropdown window.


jane@zen3:~ - Shell - Konsole <5>
Session Edit View Bookmarks Settings Help

```
            (
                { 'id'              : 'status'
                , 'name'            : 'Bridge Interface Graphs'
                , 'action'          : 'viewBridgeInterface'
                , 'permissions'     : (ZEN_VIEW, )
                },
                { 'id'              : 'perfConf'
                , 'name'            : 'Bridge Interface Template'
                , 'action'          : 'objTemplates'
                , 'permissions'     : (ZEN_CHANGE_SETTINGS, )
                },
                { 'id'              : 'viewHistory'
                , 'name'            : 'Modifications'
                , 'action'          : 'viewHistory'
                , 'permissions'     : (ZEN_VIEW, )
                },
            )
          },
        )

    isUserCreatedFlag = True

    def isUserCreated(self):
        """
        Returns the value of isUserCreated. True adds SAVE & CANCEL buttons to Details menu
        """
        return self.isUserCreatedFlag

    def viewName(self):
        """Pretty version human readable version of this object"""
        if self.RemoteAddress == '00:00:00:00:00:00' or self.Port == '-1':
            return "Unknown"
        else:
#                return self.id
            return str( self.Port ) + "-" + self.RemoteAddress

"BridgeInterface.py" [Modified][readonly] 168 lines --54%--          91,45         50%
```

*Figure 82: BridgeInterface component object file with isUserCreated defined and set to True*

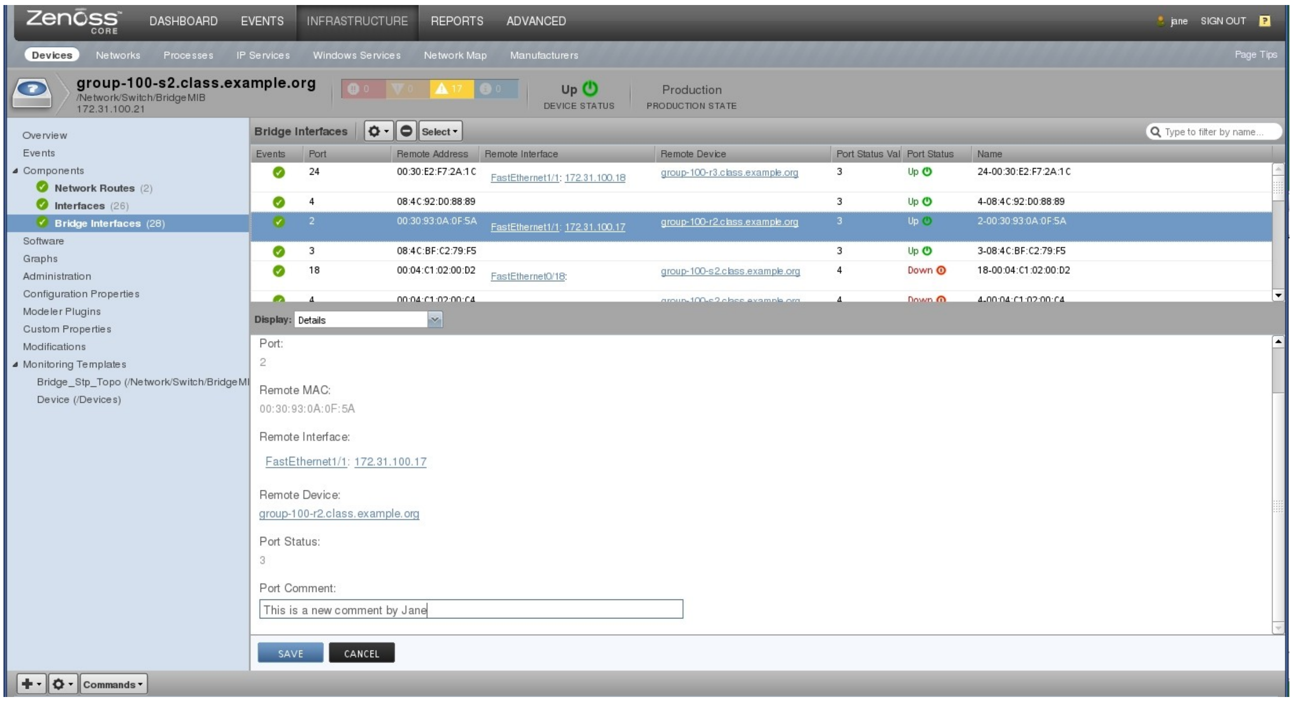
The object file BridgeInterface.py does **not** need to have an attribute explicitly set to read-write but the interfaces.py file controls which fields **are** read-write. See the added PortComment field in Figure 83. (Note that PortComment also needs adding to the BridgeInterface object file and to info.py).


jane@zen3:~ - Shell - Konsole <5>
Session Edit View Bookmarks Settings Help

```
################################################################################
#
# This program is part of the Bridge Zenpack for Zenoss.
# Copyright (C) 2010 Jane Curry
#
# This program can be used under the GNU General Public License version 2
# You can find full information here: http://www.zenoss.com/oss
#
################################################################################

__doc__="""interfaces

describes the form field to the user interface.

$Id: interfaces.py,v 1.2 2010/12/14 20:46:34 jc Exp $"""

__version__ = "$Revision: 1.4 $"[11:-2]

from Products.Zuul.interfaces import IComponentInfo
from Products.Zuul.form import schema
from Products.Zuul.utils import ZuulMessageFactory as _t


class IBridgeInterfaceInfo(IComponentInfo):
    """
Info adapter for Bridge Interface component
"""
    Port = schema.Text(title=u"Port", readonly=True, group='Details')
    RemoteAddress = schema.Text(title=u"Remote MAC", readonly=True, group='Details')
    RemoteInterface = schema.Text(title=u"Remote Interface", readonly=True,group='Details')
    RemoteDevice = schema.Text(title=u"Remote Device", readonly=True, group='Details')
    PortStatus = schema.Text(title=u"Port Status", readonly=True, group='Details')
    PortComment = schema.Text(title=u"Port Comment", group='Details')

"interfaces.py" 34 lines --2%--                                   1,54          All
```

*Figure 83: interfaces.py with an extra PortComment field that is read-write*

The result is show in Figure 84.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

0 0 17 0

Up
DEVICE STATUS

Production
PRODUCTION STATE

Bridge Interfaces

| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | Name |
|---|---|---|---|---|---|---|---|
| ✓ | 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | 3 | Up | 24-00:30:E2:F7:2A:1C |
| ✓ | 4 | 08:4C:92:D0:88:89 | | | 3 | Up | 4-08:4C:92:D0:88:89 |
| ✓ | 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | 3 | Up | 2-00:30:93:0A:0F:5A |
| ✓ | 3 | 08:4C:BF:C2:79:F5 | | | 3 | Up | 3-08:4C:BF:C2:79:F5 |
| ✓ | 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | 4 | Down | 18-00:04:C1:02:00:D2 |
| ✓ | 4 | 00:04:C1:02:00:C4 | | group-100-s2.class.example.org | 4 | Down | 4-00:04:C1:02:00:C4 |

Display: Details

Port:
2

Remote MAC:
00:30:93:0A:0F:5A

Remote Interface:
FastEthernet1/1: 172.31.100.17

Remote Device:
group-100-r2.class.example.org

Port Status:
3

Port Comment:
This is a new comment by Jane

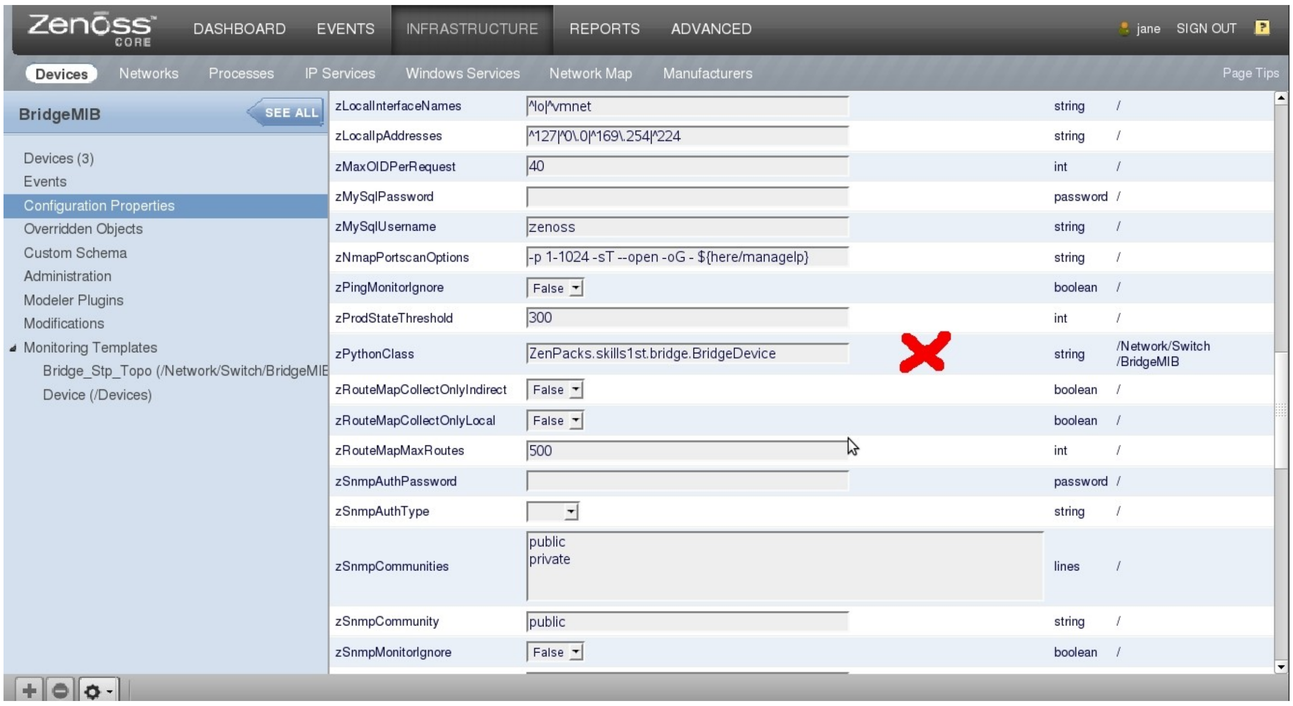
*Figure 84: Details dropdown window for a component with editable Port Comment field*

### 4.3.10 Linking development mode elements with source mode elements

At this stage we have:

- A new device class, *BridgeMIB*, a subclass of /Devices/Network/Switch, created via the GUI and added to the ZenPack in Development mode
- Some MIBs added to the ZenPack in Development mode
- Two new object class files, *BridgeDevice.py* and *BridgeInterface.py* in the base directory of the ZenPack, created in source mode
- Two modeler plugins, *BridgeInterfaceMib.py* and *BridgeDevice.py* in the modeler/plugins subdirectory of the base ZenPack directory, created in source mode
- Two skins files, *BridgeDeviceDetail.pt* and *viewBridgeInterface.pt* in the skins/ZenPacks.skills-1st.bridge subdirectory of the base ZenPack directory, created in source mode. Both are necessary for Zenoss 2; BridgeDeviceDetail.pt is redundant for Zenoss 3 unless a standalone Bridge Interfaces left-hand menu is required. viewBridgeInterface.pt is required for Zenoss 3 to provide *Bridge Interface Graphs* from the Display dropdown menu.
- An *info.py, interfaces.py* and *configure.zcml* in the base ZenPack directory and a *resources* subdirectory containing *bridge.js* to provide a custom device component panel for Zenoss 3, created in source mode

Nothing yet links the new device class with the object classes and their associated modelers. This is achieved using the GUI.

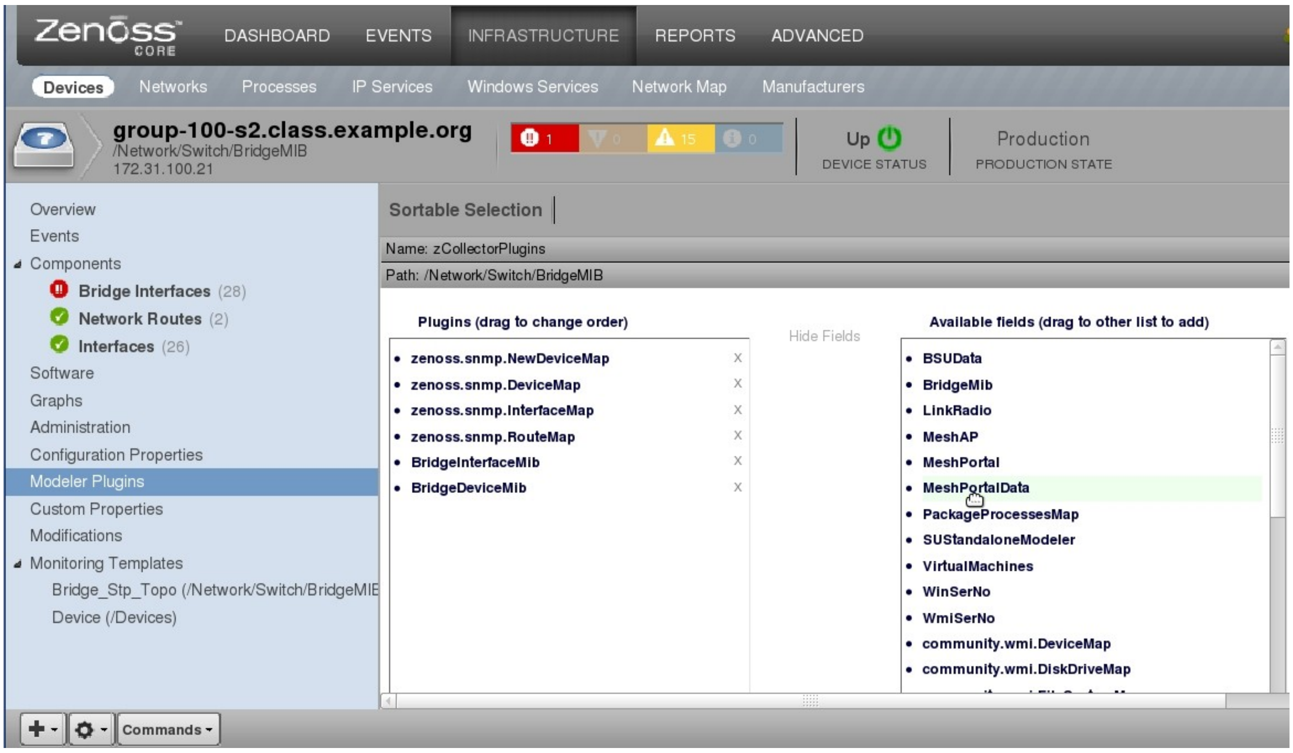


| | | | |
|---|---|---|---|
| zLocalInterfaceNames | ^lo\|^vmnet | string | / |
| zLocalIpAddresses | ^127\|^0\.0\|^169\.254\|^224 | string | / |
| zMaxOIDPerRequest | 40 | int | / |
| zMySqlPassword | | password | / |
| zMySqlUsername | zenoss | string | / |
| zNmapPortscanOptions | -p 1-1024 -sT --open -oG - ${here/manageIp} | string | / |
| zPingMonitorIgnore | False | boolean | / |
| zProdStateThreshold | 300 | int | / |
| zPythonClass | ZenPacks.skills1st.bridge.BridgeDevice | string | /Network/Switch /BridgeMIB |
| zRouteMapCollectOnlyIndirect | False | boolean | / |
| zRouteMapCollectOnlyLocal | False | boolean | / |
| zRouteMapMaxRoutes | 500 | int | / |
| zSnmpAuthPassword | | password | / |
| zSnmpAuthType | | string | / |
| zSnmpCommunities | public private | lines | / |
| zSnmpCommunity | public | string | / |
| zSnmpMonitorIgnore | False | boolean | / |

*Figure 85: Linking a device class with the object class file that describes its unique properties*

To associate the device class, *BridgeMIB*, with the object class *BridgeDevice*, simply modify the zProperty **zPythonClass,** either for an individual device or for a subclass of devices. For a device class, you need to click the *DETAILS* link at the top of the left-hand menu to access the *Configuration Properties*. Remember to save the modifications. The zPythonClass should be the fully-qualified *object* name. In this case, the object is defined in this ZenPack so the zPythonClass is *ZenPacks.skills1st.bridge.BridgeDevice* (no .py on the end). There is no direct association here with the BridgeInterface class as that is a contained component object class of BridgeDevice.

The second link required, is between the device class and the modeler plugins to be deployed for that class. This is done from the *More -> Collector Plugins* menu of either an individual device or a device class, for Zenoss 2, or in Zenoss 3 from the same *DETAILS* link for a device class, or from a device's *Configuration Properties* menu. If the plugin source code is valid then the name of the plugin should automatically appear in the *Add Fields* list. Required plugins are dragged to the selected area and can be reordered simply by dragging them around. Again, don't forget the *Save* button.



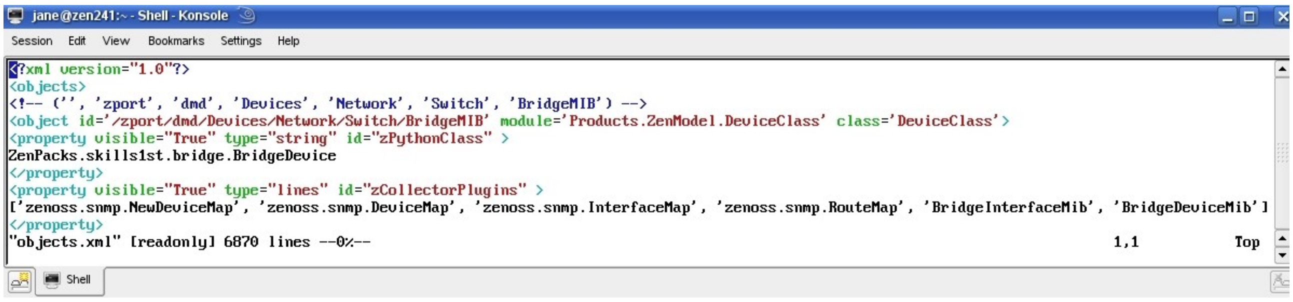
Name: zCollectorPlugins
Path: /Network/Switch/BridgeMIB

Plugins (drag to change order)
- zenoss.snmp.NewDeviceMap
- zenoss.snmp.DeviceMap
- zenoss.snmp.InterfaceMap
- zenoss.snmp.RouteMap
- BridgeInterfaceMib
- BridgeDeviceMib

Available fields (drag to other list to add)
- BSUData
- BridgeMib
- LinkRadio
- MeshAP
- MeshPortal
- MeshPortalData
- PackageProcessesMap
- SUStandaloneModeler
- VirtualMachines
- WinSerNo
- WmiSerNo
- community.wmi.DeviceMap
- community.wmi.DiskDriveMap

*Figure 86: Associating a device with a set of modeler plugins*

To propagate these associations to the ZenPack, navigate to *ADVANCED -> Settings -> ZenPacks*, select the ZenPack, and use the *Action* icon at the bottom of the left-hand menu to *Export ZenPack*. In addition to creating the egg file ( *ZenPacks.skills1st.bridge-1.0.4-py2.6.egg* ) for the ZenPack in *$ZENHOME/export*, exporting also updates the *object.xml* file in the *objects* subdirectory of the ZenPack.



```
<?xml version="1.0"?>
<objects>
<!-- ('', 'zport', 'dmd', 'Devices', 'Network', 'Switch', 'BridgeMIB') -->
<object id='/zport/dmd/Devices/Network/Switch/BridgeMIB' module='Products.ZenModel.DeviceClass' class='DeviceClass'>
<property visible="True" type="string" id="zPythonClass" >
ZenPacks.skills1st.bridge.BridgeDevice
</property>
<property visible="True" type="lines" id="zCollectorPlugins" >
['zenoss.snmp.NewDeviceMap', 'zenoss.snmp.DeviceMap', 'zenoss.snmp.InterfaceMap', 'zenoss.snmp.RouteMap', 'BridgeInterfaceMib', 'BridgeDeviceMib']
</property>
"objects.xml" [readonly] 6870 lines --0%--                                                    1,1           Top
```

*Figure 87: Start of objects.xml showing zPythonClass and zCollectorPlugins*

The egg file is created by first copying the directory hierarchy to the *build/lib* subdirectory. For example, if the ZenPack code is in */jane/ZenPacks.skills1st.bridge* then the subdirectory hierarchy starts with the *ZenPacks* directory. At the same level, under */jane/ZenPacks.skills1st.bridge*, a *build* directory will be created or updated in which is a *lib* directory and the hierarchy starting from the ZenPacks directory is copied here - */jane/ZenPacks.skills1st.bridge/build/lib/ZenPacks*. The egg file is actually constructed from this build subtree.

Note that with Zenoss 3.0.3, there is currently a bug whereby the *build/lib* subdirectory is **not** cleaned out before export. This means that existing files **will** be updated, new files **will** be added but if old files have been deleted then they will **still exist** in the *build/lib* subdirectory and hence, in the new egg file. This is documented in ticket 7324 ( http://dev.zenoss.com/trac/ticket/7324 ). If files have been deleted from ZenPack sources, ensure that everything under the *build/lib* subdirectory is removed before exporting the ZenPack.

Once an egg file has been created in the *export* subdirectory, it can be moved to a different system and loaded there.

# 5 Gathering Performance Data

Performance data is gathered, usually by either the *zenperfsnmp* daemon (for SNMP data), or by the *zencommand* daemon (for ssh data). Other performance data collectors may be made available by other ZenPacks.

By default, zenperfsnmp runs every 5 minutes; for ssh-collected data, the performance template allows you to specify a collection interval although zencommand only runs every minute, by default. With Zenoss Core, a single zenperfsnmp daemon is available (although it is possible to deploy others); with Zenoss Enterprise, multiple data collectors can be configured fairly easily. This means in a typical Zenoss Core installation, that there really is only one polling interval configuration to collect SNMP performance data. The default of 5 minutes can be changed easily using the *ADVANCED -> Collectors -> localhost -> Edit* menu, but there is still only one collector (or *monitor*, as it used to be called).

To specify performance data for collection, Zenoss **templates** need to be created and **bound** to a device or device class. A template defines:

- One or more data sources
- One or more data points
- Threshold values, if required
- Graph definitions, if required

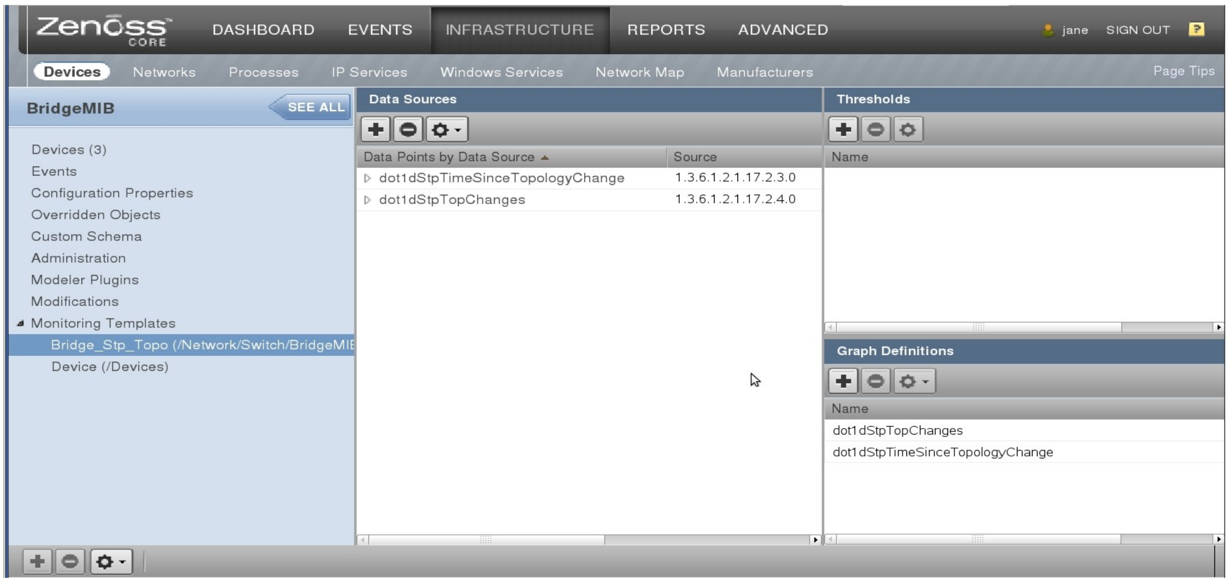
## 5.1 Performance templates for devices

For the Bridge MIB ZenPack, some data may be required pertinent to the whole device; other data will be per-port. Device-wide data can be gathered in the usual manner and will be displayed under the standard *Perf* tab for Zenoss 2 or the left-hand *Graphs* menu for Zenoss 3.

For example, the Bridge MIB provides values out of the Spanning Tree Protocol (Stp) subtree of the MIB which gives:

| | | |
|---|---|---|
| • dot1dStpTimeSinceTopologyChange | (TimeTicks) | .1.3.6.1.2.1.17.2.3.0 |
| • dot1dStpTopChanges | (Counter32) | .1.3.6.1.2.1.17.2.4.0 |

These values give a measure of the number of centi-seconds since the last Stp topology change and the number of topology changes since the last initialise or reboot of the device. Note that both have *.0* on the end – they are scalar MIB values ie. there is only one value for the whole device. A Zenoss template can be configured to collect and graph these values.



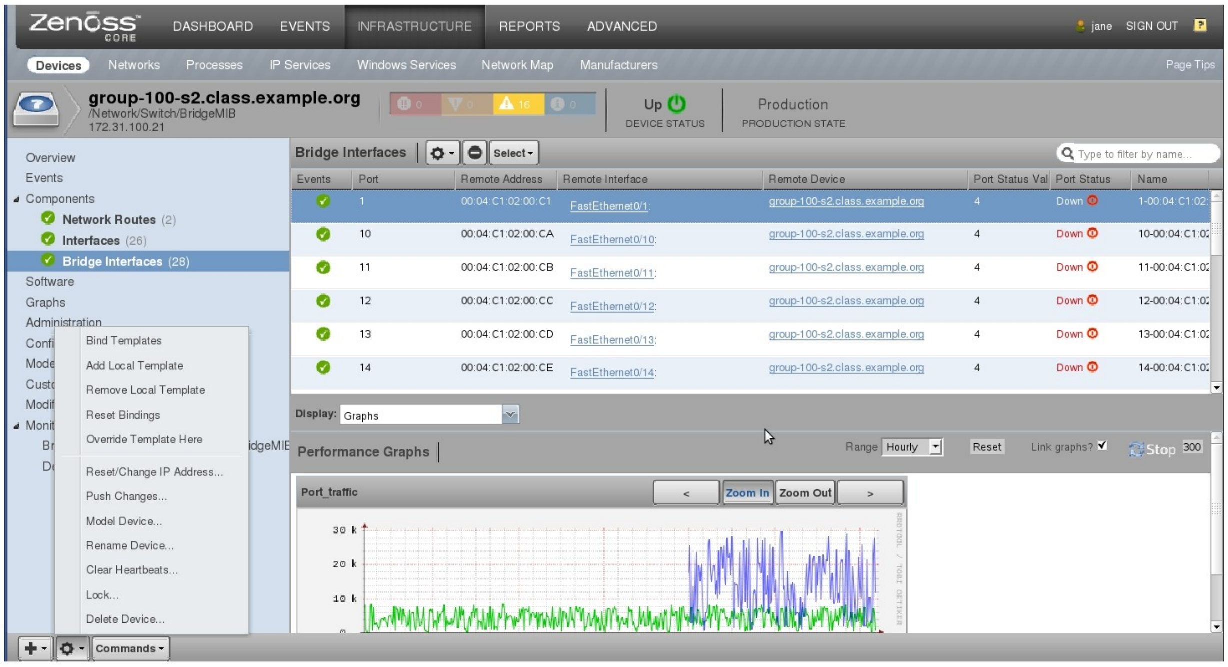
| Data Points by Data Source | Source |
|---|---|
| dot1dStpTimeSinceTopologyChange | 1.3.6.1.2.1.17.2.3.0 |
| dot1dStpTopChanges | 1.3.6.1.2.1.17.2.4.0 |

| Name |
|---|
| dot1dStpTopChanges |
| dot1dStpTimeSinceTopologyChange |

*Figure 88: Zenoss performance template to gather Bridge Stp topology change data for the BridgeMIB device class*

Templates is an area where Zenoss 3 has made great improvements in the clarity of the GUI. With Zenoss 2, it was not always easy to see whether a template was **bound** to a device or device class; with Zenoss 3, a device class has a *Monitoring Templates* left-hand menu from its *DETAILS* link (as seen in Figure 88). The submenus show each bound template. Similarly, a device also has a *Monitoring Templates* menu.

To activate the template, it must be **bound** either to a device class or to a specific device. Use the *Bind Templates* menu from the *Action* icon to bind a template to a specific device.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

Up — DEVICE STATUS
Production — PRODUCTION STATE

| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | Name |
|---|---|---|---|---|---|---|---|
| ✓ | 1 | 00:04:C1:02:00:C1 | FastEthernet0/1: | group-100-s2.class.example.org | 4 | Down | 1-00:04:C1:02 |
| ✓ | 10 | 00:04:C1:02:00:CA | FastEthernet0/10: | group-100-s2.class.example.org | 4 | Down | 10-00:04:C1:0 |
| ✓ | 11 | 00:04:C1:02:00:CB | FastEthernet0/11: | group-100-s2.class.example.org | 4 | Down | 11-00:04:C1:0 |
| ✓ | 12 | 00:04:C1:02:00:CC | FastEthernet0/12: | group-100-s2.class.example.org | 4 | Down | 12-00:04:C1:0 |
| ✓ | 13 | 00:04:C1:02:00:CD | FastEthernet0/13: | group-100-s2.class.example.org | 4 | Down | 13-00:04:C1:0 |
| ✓ | 14 | 00:04:C1:02:00:CE | FastEthernet0/14: | group-100-s2.class.example.org | 4 | Down | 14-00:04:C1:0 |

*Figure 89: Bind Templates menu from Action icon for a specific device*

To create templates, use the *ADVANCED -> Monitoring Templates* option and use the "+" icon at the bottom of the left-hand menu to *Add a Monitoring Template*; you will be prompted for the name, and the device class in the *Template Path* field.

The *Monitoring Templates* dialog  can display existing templates either by *Template* or by *Device Class* and also indicates with an icon whether a template is a component template and whether it is bound to a particular device class.  The bound status of a device class template can be "toggled" from the *Action* icon *Toggle Template Binding* menu.

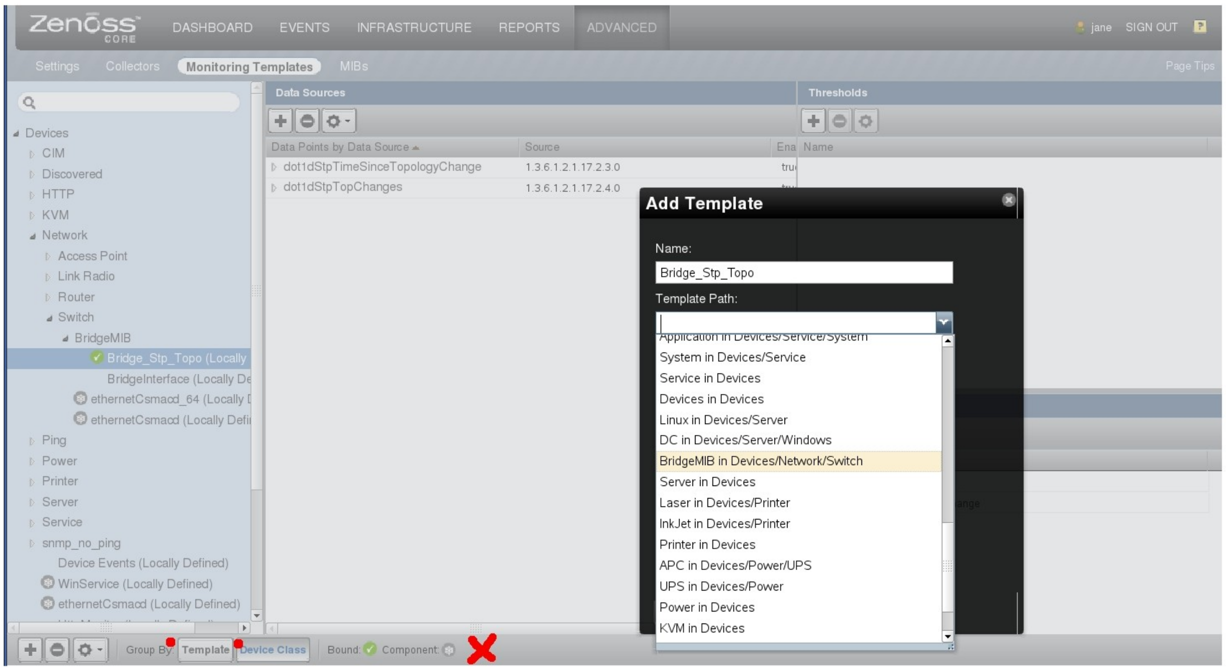


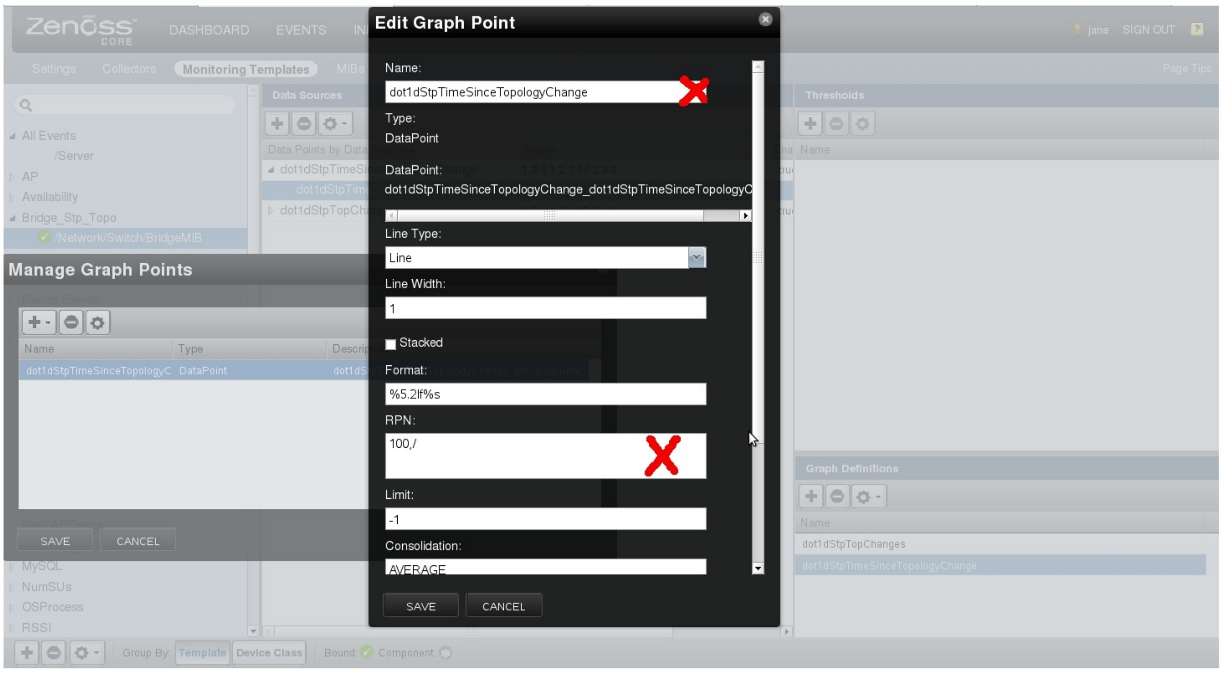
*Figure 90: Add template menu highlighting template "Group by" and template type*

Once a template has been created and bound, you should then see graph outlines under the *Perf* tab (Zenoss 2) or *Graphs* menu (Zenoss 3) on the device's detailed page; however it will typically be two zenperfsnmp collection intervals before you start to see data.

Note that since dot1dStpTimeSinceTopologyChange is in units of centi-seconds the graph point has been modified with a Reverse Polish Notation (RPN) expression to divide by 100 ( *100,/* ) to convert it to seconds and the *Units* field of the graph definition has been set to *secs*.

Also note with Zenoss 3.0.3 there is a small bug whereby when editing graph points the data source name is omitted from the dialog and no changes can be saved until the name is re-added. This is documented in ticket 7597 ( http://dev.zenoss.org/trac/ticket/7597 ).

Also note that, although on the Graph Definition the *Has Summary* box is ticked by default, you may not see cur, avg and max values for the data at the bottom of the graph. This is a known issue with Zenoss 3.0.2 / 3 and Firefox whereby this field keeps resetting to being unticked when the *Manage Graph Points -> Edit Graph Point* menu sequence has been used. It is documented as ticket 7404 , with a workaround using the Zope Management Interface (ZMI), at http://dev.zenoss.com/trac/ticket/7404 and a patch is now available.



Name:
dot1dStpTimeSinceTopologyChange

Type:
DataPoint

DataPoint:
dot1dStpTimeSinceTopologyChange_dot1dStpTimeSinceTopologyC

Line Type:
Line

Line Width:
1

Stacked

Format:
%5.2lf%s

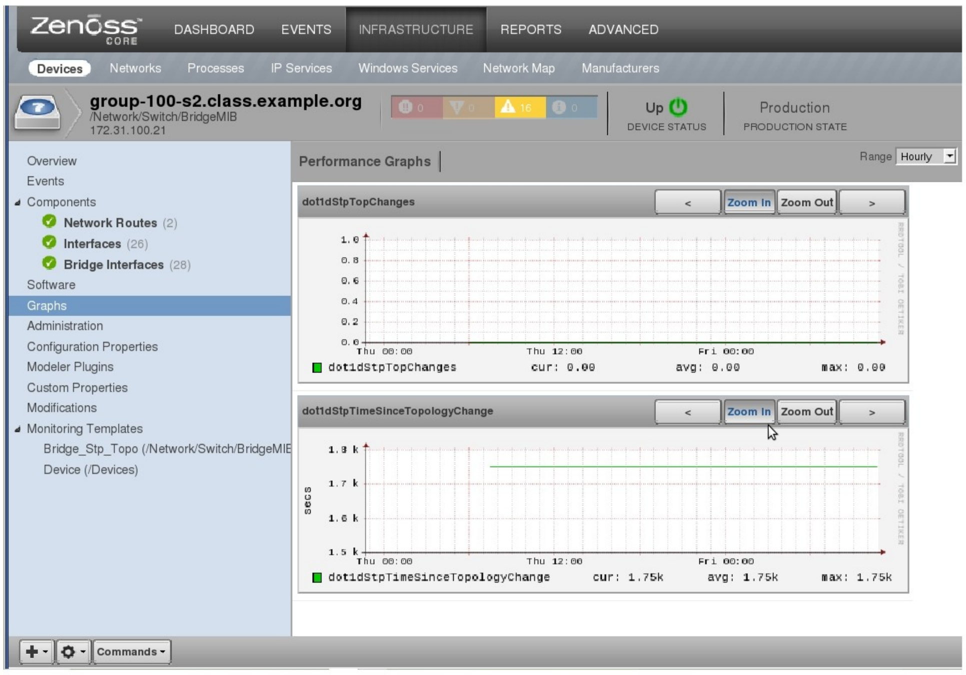
RPN:
100,/

Limit:
-1

Consolidation:
AVERAGE

*Figure 91: dot1d Stp template showing Reverse Polish Notation (RPN) to change data units*

The resultant graph is shown in Figure 92.



```
dot1dStpTopChanges                 cur: 0.00      avg: 0.00      max: 0.00
dot1dStpTimeSinceTopologyChange    cur: 1.75k     avg: 1.75k     max: 1.75k
```

*Figure 92: Performance graph for switch showing dot1d Stp data*

Templates to be included with the ZenPack should be added using the *Add to ZenPack* menu from the *Action* icon, as shown in Figure 93.

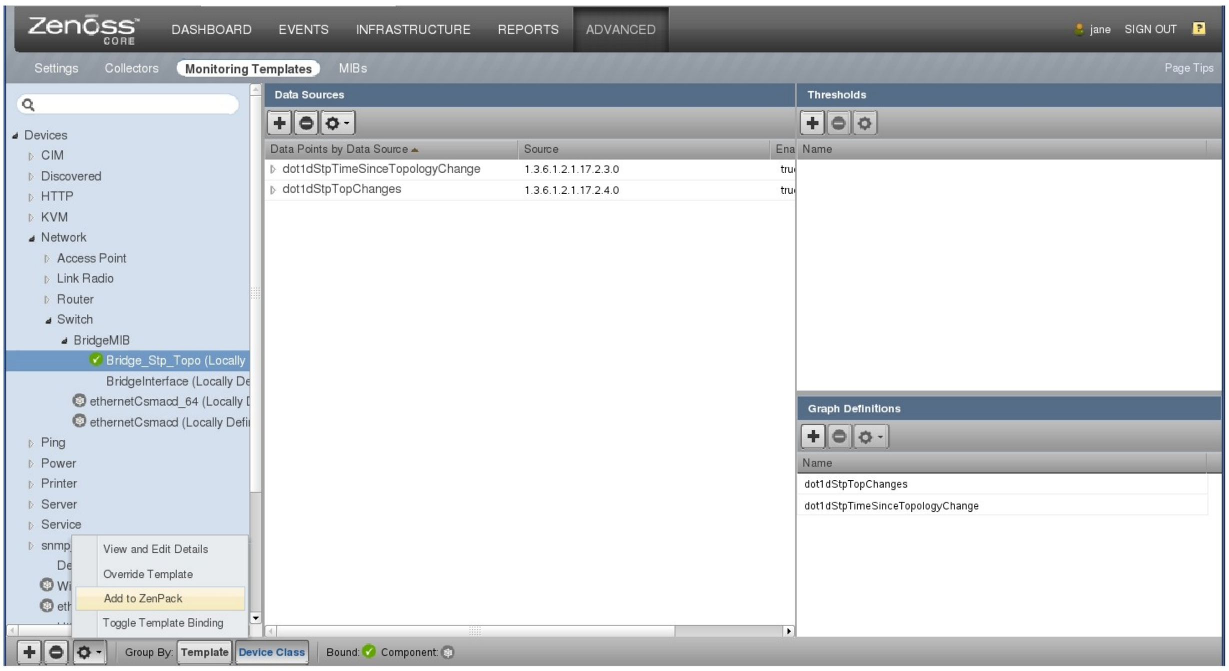


Figure 93: *Adding Zenoss performance Template to a ZenPack*

Re-exporting the ZenPack will also update the definition of the BridgeMIB device class in *objects/objects.xml*, including the *zDeviceTemplates* zProperty, if you have bound the template to that device class.

## 5.2 Performance templates for contained devices

To get performance information for the switch **ports** that are supported by the ZenPack, there are two important factors:

- A Zenoss Template **with exactly the same name as a contained, component class object**, will **automatically** be bound to instances of that object. The object class representing a switch port is BridgeInterface; thus a template called BridgeInterface will automatically be bound to such objects.
- When specifying a template for SNMP performance data to be collected, unless the data is a scalar, you do not specify the instance to be collected. The instances are taken from the object class (BridgeInterface) **snmpindex** attribute.

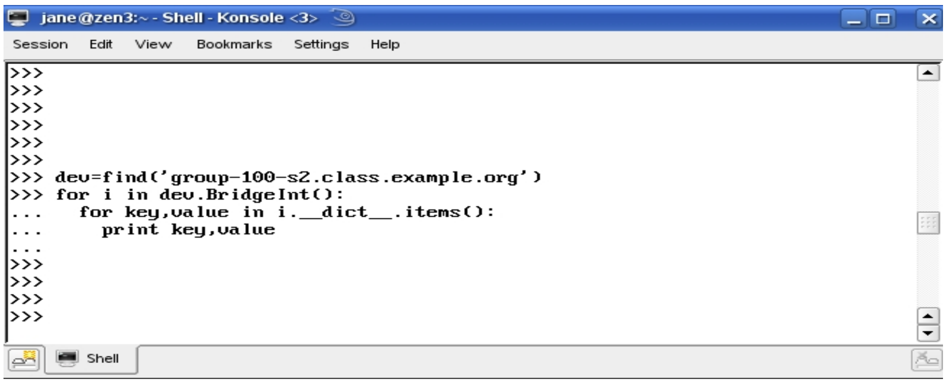
Remember when the BridgeInterfaceMib modeler plugin was created, it populated not only the unique attributes of the BridgeInterface object class, but also populated the inherited attributes of:

- id

- snmpindex

Since most of the useful performance data from the BRIDGE MIB is indexed by the Port value, the snmpindex attribute was set to this value, having first converted the raw data to an integer type. Thus for a Catalyst 2900, the Port values, and hence the snmpindex values, run from 13 to 38.

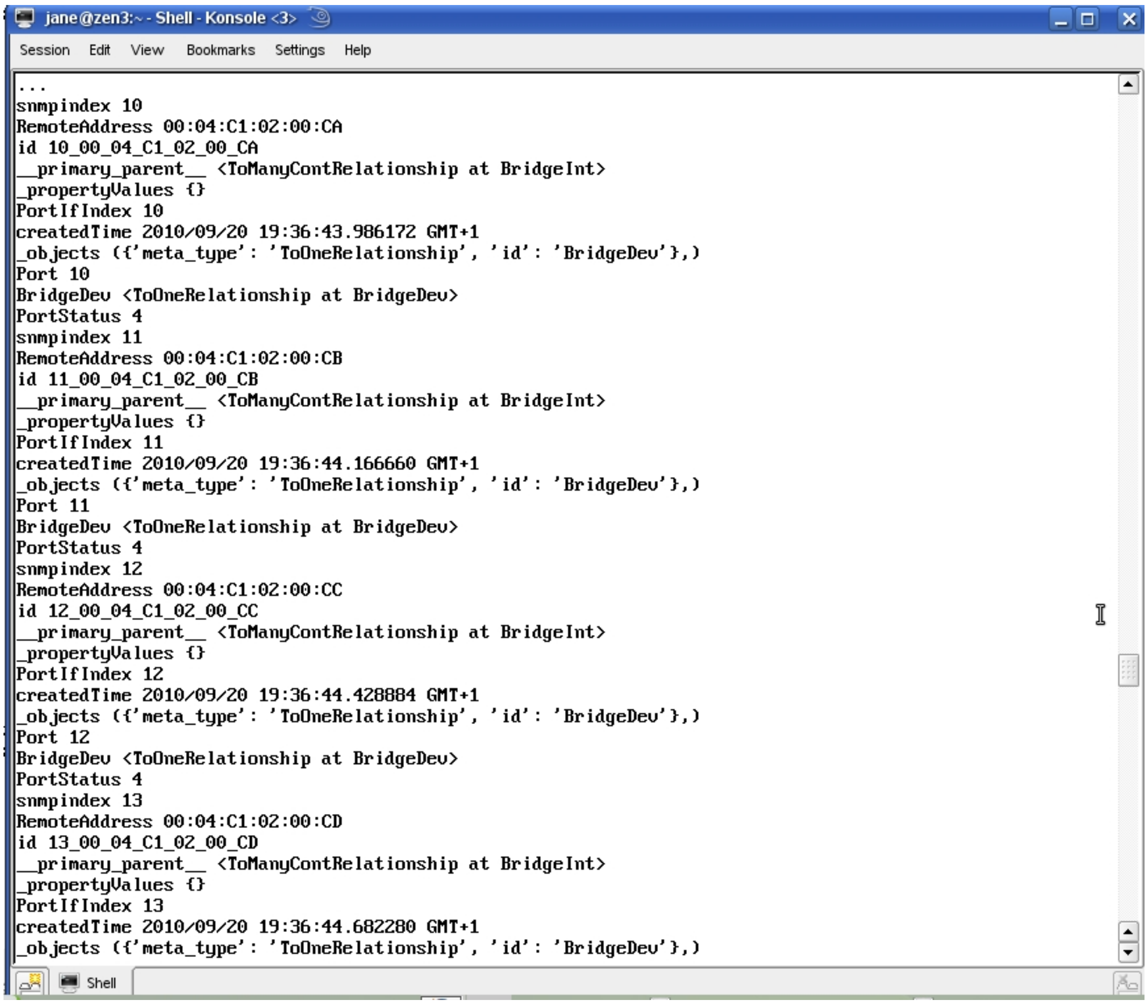
The Zenoss *zendmd* utility is useful to see the values of objects – see Figure 94 for the code and Figure 95 for an output fragment.



```
>>>
>>>
>>>
>>>
>>>
>>>
>>> dev=find('group-100-s2.class.example.org')
>>> for i in dev.BridgeInt():
...   for key,value in i.__dict__.items():
...     print key,value
...
>>>
>>>
>>>
>>>
```

*Figure 94: Using zendmd to see values of the attributes of a BridgeInterface object*

Note in Figure 94 that you start with a device and then print information for the BridgeInt **relationship** for that device.


jane@zen3:~ - Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
Shell

```
...
snmpindex 10
RemoteAddress 00:04:C1:02:00:CA
id 10_00_04_C1_02_00_CA
__primary_parent__ <ToManyContRelationship at BridgeInt>
_propertyValues {}
PortIfIndex 10
createdTime 2010/09/20 19:36:43.986172 GMT+1
_objects ({'meta_type': 'ToOneRelationship', 'id': 'BridgeDev'},)
Port 10
BridgeDev <ToOneRelationship at BridgeDev>
PortStatus 4
snmpindex 11
RemoteAddress 00:04:C1:02:00:CB
id 11_00_04_C1_02_00_CB
__primary_parent__ <ToManyContRelationship at BridgeInt>
_propertyValues {}
PortIfIndex 11
createdTime 2010/09/20 19:36:44.166660 GMT+1
_objects ({'meta_type': 'ToOneRelationship', 'id': 'BridgeDev'},)
Port 11
BridgeDev <ToOneRelationship at BridgeDev>
PortStatus 4
snmpindex 12
RemoteAddress 00:04:C1:02:00:CC
id 12_00_04_C1_02_00_CC
__primary_parent__ <ToManyContRelationship at BridgeInt>
_propertyValues {}
PortIfIndex 12
createdTime 2010/09/20 19:36:44.428884 GMT+1
_objects ({'meta_type': 'ToOneRelationship', 'id': 'BridgeDev'},)
Port 12
BridgeDev <ToOneRelationship at BridgeDev>
PortStatus 4
snmpindex 13
RemoteAddress 00:04:C1:02:00:CD
id 13_00_04_C1_02_00_CD
__primary_parent__ <ToManyContRelationship at BridgeInt>
_propertyValues {}
PortIfIndex 13
createdTime 2010/09/20 19:36:44.682280 GMT+1
_objects ({'meta_type': 'ToOneRelationship', 'id': 'BridgeDev'},)
```

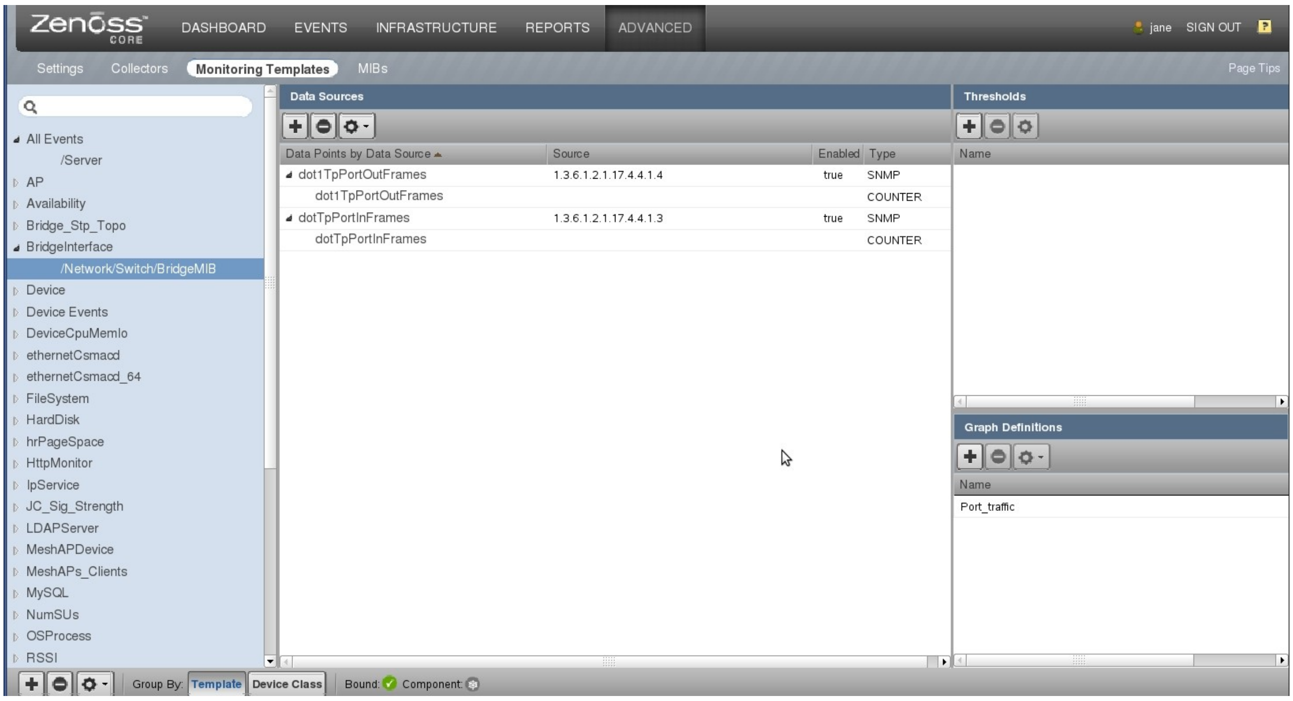
*Figure 95: Output of the zendmd code to view attributes of a BridgeInterface object*

The BridgeInterfaceMib modeler plugin set the *id* attribute of a BridgeInterface object by concatenating the Port number, an underscore and the RemoteAddress. The Python *prepId* function was applied to ensure uniqueness. An example would be *10_00_04_C1_02_00_CA.*

The BRIDGE MIB provides values for:

| | | |
|---|---|---|
| • dot1TpPortInFrames | (Counter32) | .1.3.6.1.2.1.17.4.4.1.3 |
| • dot1TpPortInFrames | (Counter32) | .1.3.6.1.2.1.17.4.4.1.4 |

These are performance counters for traffic seen on a transparent bridge port and they are indexed by port number. To be able to graph these values per-port, when an individual port is clicked on in the GUI, create a template with the name *BridgeInterface* for the /Devices/Network/Switch/BridgeMIB device class.

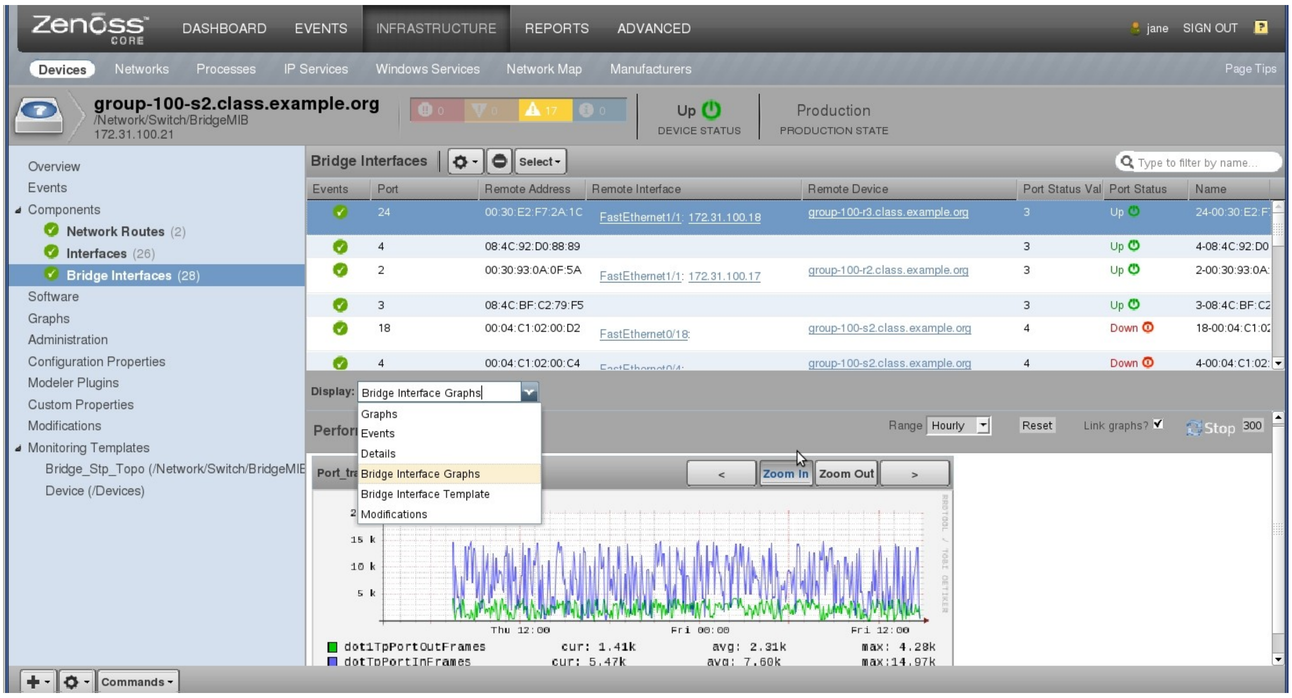


| Data Points by Data Source | Source | Enabled | Type |
|---|---|---|---|
| dot1TpPortOutFrames | 1.3.6.1.2.1.17.4.4.1.4 | true | SNMP |
| dot1TpPortOutFrames | | | COUNTER |
| dotTpPortInFrames | 1.3.6.1.2.1.17.4.4.1.3 | true | SNMP |
| dotTpPortInFrames | | | COUNTER |

| Name |
|---|
| Port_traffic |

Figure 96: BridgeInterface template

There is no need to bind this template to any device or device class. To see performance data for a device, simply click on a port under the *Components* submenu for *Bridge Interfaces* and select either the *Graphs* or *Bridge Interface Graphs* Display dropdown (remembering that it will generally take two SNMP polling intervals before data is displayed).



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

0 0 17 0

Up DEVICE STATUS

Production PRODUCTION STATE

| Events | Port | Remote Address | Remote Interface | Remote Device | Port Status Val | Port Status | Name |
|---|---|---|---|---|---|---|---|
| ✓ | 24 | 00:30:E2:F7:2A:1C | FastEthernet1/1: 172.31.100.18 | group-100-r3.class.example.org | 3 | Up | 24-00:30:E2:F |
| ✓ | 4 | 08:4C:92:D0:88:89 | | | 3 | Up | 4-08:4C:92:D0 |
| ✓ | 2 | 00:30:93:0A:0F:5A | FastEthernet1/1: 172.31.100.17 | group-100-r2.class.example.org | 3 | Up | 2-00:30:93:0A: |
| ✓ | 3 | 08:4C:BF:C2:79:F5 | | | 3 | Up | 3-08:4C:BF:C2 |
| ✓ | 18 | 00:04:C1:02:00:D2 | FastEthernet0/18: | group-100-s2.class.example.org | 4 | Down | 18-00:04:C1:02 |
| ✓ | 4 | 00:04:C1:02:00:C4 | FastEthernet0/4: | group-100-s2.class.example.org | 4 | Down | 4-00:04:C1:02: |

Display: Bridge Interface Graphs

```
🟩 dot1TpPortOutFrames     cur: 1.41k      avg: 2.31k      max: 4.28k
🟦 dotTpPortInFrames      cur: 5.47k      avg: 7.60k      max:14.97k
```

Figure 97: Performance graph for a specific switch port

The bottom three options on the *Display* menu shown in Figure 97 were defined in the object class file BridgeInterface.py as shown in Figure 34 on page 46. The skins file to display performance graphs for a port is in *skins/ZenPacks.skills-1st.bridge/viewBridgeInterface.pt* shown in Figure 53 on page 69.

Remember to add the BridgeInterface performance template to the ZenPack when it is complete and to re-export the ZenPack.

# 6 Testing and debugging ZenPacks

The chances of getting a ZenPack with new source code, correct first time, is not high. This section offers some testing and debugging hints.

## 6.1 Testing

There may be four main areas where you have added code; object class files, modeler plugins, skins and JavaScript files.

### 6.1.1 Testing new object class files

If you have created or changed object class files, you should always delete any discovered instances that use those files and rediscover them to ensure that any relationship changes are established correctly. You should certainly recycle **zenhub** and **zopectl** with:

- zenhub restart
- zopectl restart

Typically you will be doing initial testing with a single device so delete the device and use the *Add Device* menu to re-add it, ensuring that you specify your new device class in the *Device Class* dropdown. Adding the device runs *zendisc* which calls *zenmodeler*. You may see error messages in the discovery GUI. Usually they are quite good at pinpointing the problem to a particular line in a particular file. Watch out especially for syntax errors in your code such as missing closing brackets, missing quotes or missing colons ( : ).

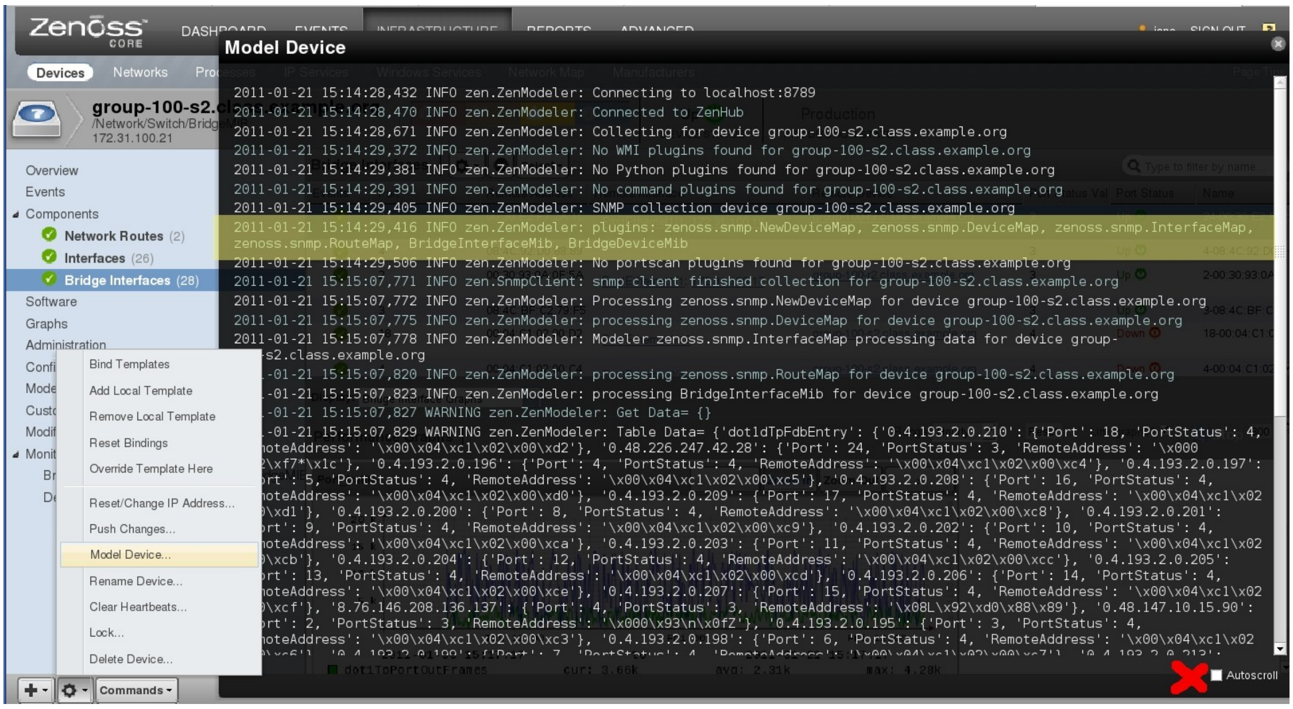
Another way to start testing object class files is to use the Zope ZMI interface to navigate to *http://zen3.class.example.org:8080/zport/dmd/manage* and then navigate down Devices/Network/Switch/BridgeMib/devices/<a specific device> and check that the BridgeInt relationship exists.

A classic error to make in Python files is to get white space indentation wrong. Python uses indentation to structure *if*, *while*, *for* and other constructs; you must be consistent with the number of spaces used at each level of indentation.

## 6.1.2 Testing modeler plugins

If you have created or changed a modeler plugin, you need to restart **zenhub** and **zopectl**; typically you do **not** need to delete your test device and re-add it. It should be sufficient to simply use the *Model Device* menu from the *Action* icon and watch the output (note that you do need Zenoss 3.0.3 or you may find a bug causes the modeler output to be invisible!).

Note the dialogue particularly to ensure that your modeler does at least attempt to run – the output will show what plugins are to be run. You may need to uncheck the Autoscroll box at the bottom-right to be able to scroll back up the window.

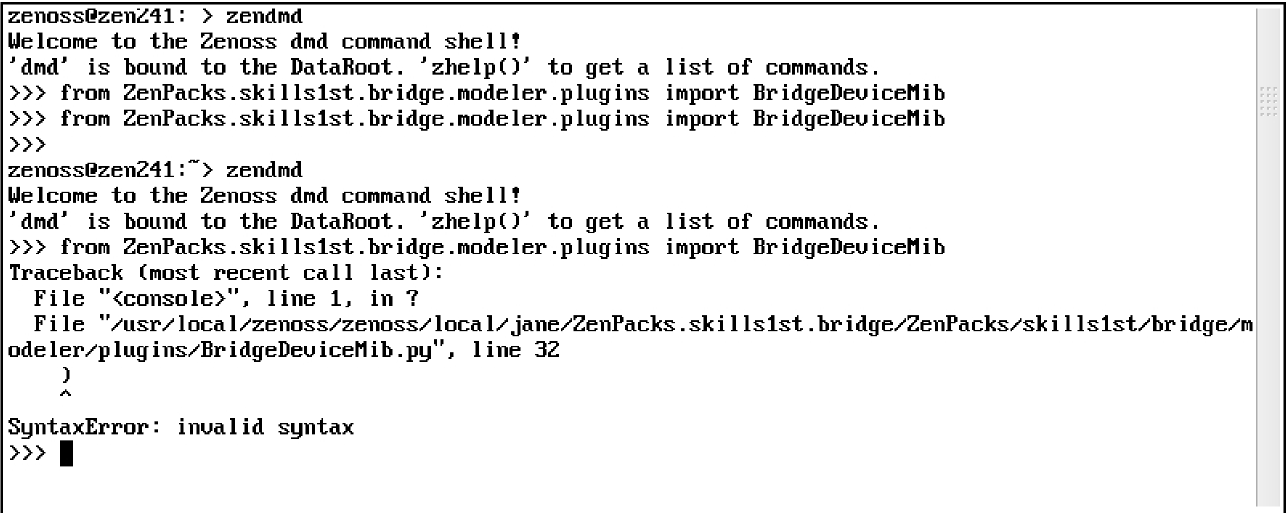


```
2011-01-21 15:14:28,432 INFO zen.ZenModeler: Connecting to localhost:8789
2011-01-21 15:14:28,470 INFO zen.ZenModeler: Connected to ZenHub
2011-01-21 15:14:28,671 INFO zen.ZenModeler: Collecting for device group-100-s2.class.example.org
2011-01-21 15:14:29,372 INFO zen.ZenModeler: No WMI plugins found for group-100-s2.class.example.org
2011-01-21 15:14:29,381 INFO zen.ZenModeler: No Python plugins found for group-100-s2.class.example.org
2011-01-21 15:14:29,391 INFO zen.ZenModeler: No command plugins found for group-100-s2.class.example.org
2011-01-21 15:14:29,405 INFO zen.ZenModeler: SNMP collection device group-100-s2.class.example.org
2011-01-21 15:14:29,416 INFO zen.ZenModeler: plugins: zenoss.snmp.NewDeviceMap, zenoss.snmp.DeviceMap, zenoss.snmp.InterfaceMap,
zenoss.snmp.RouteMap, BridgeInterfaceMib, BridgeDeviceMib
2011-01-21 15:14:29,506 INFO zen.ZenModeler: No portscan plugins found for group-100-s2.class.example.org
2011-01-21 15:15:07,771 INFO zen.SnmpClient: snmp client finished collection for group-100-s2.class.example.org
2011-01-21 15:15:07,772 INFO zen.ZenModeler: Processing zenoss.snmp.NewDeviceMap for device group-100-s2.class.example.org
2011-01-21 15:15:07,775 INFO zen.ZenModeler: processing zenoss.snmp.DeviceMap for device group-100-s2.class.example.org
2011-01-21 15:15:07,778 INFO zen.ZenModeler: Modeler zenoss.snmp.InterfaceMap processing data for device group-
-s2.class.example.org
-01-21 15:15:07,820 INFO zen.ZenModeler: processing zenoss.snmp.RouteMap for device group-100-s2.class.example.org
-01-21 15:15:07,823 INFO zen.ZenModeler: processing BridgeInterfaceMib for device group-100-s2.class.example.org
-01-21 15:15:07,827 WARNING zen.ZenModeler: Get Data= {}
-01-21 15:15:07,829 WARNING zen.ZenModeler: Table Data= {'dot1dTpFdbEntry': {'0.4.193.2.0.210': {'Port': 18, 'PortStatus': 4,
oteAddress': '\x00\x04\xc1\x02\x00\xd2'}, '0.48.226.247.42.28': {'Port': 24, 'PortStatus': 3, 'RemoteAddress': '\x000
\xf7*\x1c'}, '0.4.193.2.0.196': {'Port': 4, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02\x00\xc4'}, '0.4.193.2.0.197':
rt': 5, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02\x00\xc5'}, '0.4.193.2.0.208': {'Port': 16, 'PortStatus': 4,
oteAddress': '\x00\x04\xc1\x02\x00\xd0'}, '0.4.193.2.0.209': {'Port': 17, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02
\xd1'}, '0.4.193.2.0.200': {'Port': 8, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02\x00\xc8'}, '0.4.193.2.0.201':
rt': 9, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02\x00\xc9'}, '0.4.193.2.0.202': {'Port': 10, 'PortStatus': 4,
oteAddress': '\x00\x04\xc1\x02\x00\xca'}, '0.4.193.2.0.203': {'Port': 11, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02
\xcb'}, '0.4.193.2.0.204': {'Port': 12, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02\x00\xcc'}, '0.4.193.2.0.205':
rt': 13, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02\x00\xcd'}, '0.4.193.2.0.206': {'Port': 14, 'PortStatus': 4,
oteAddress': '\x00\x04\xc1\x02\x00\xce'}, '0.4.193.2.0.207': {'Port': 15, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02
\xcf'}, '8.76.146.208.136.137': {'Port': 4, 'PortStatus': 3, 'RemoteAddress': '\x08L\x92\xd0\x88\x89'}, '0.48.147.10.15.90':
rt': 2, 'PortStatus': 3, 'RemoteAddress': '\x000\x93\n\x0fZ'}, '0.4.193.2.0.195': {'Port': 3, 'PortStatus': 4,
oteAddress': '\x00\x04\xc1\x02\x00\xc3'}, '0.4.193.2.0.198': {'Port': 6, 'PortStatus': 4, 'RemoteAddress': '\x00\x04\xc1\x02
```

*Figure 98: Output from Model Device highlighting the plugins to be run*

If your modeler doesn't appear on the plugins list it is probably a compilation error. Remember that you have created python source files ( ending in *.py*); Zenoss will compile-on-demand to generate *.pyc* files. A good check is always to inspect the base Zenoss directory and the modeler/plugins directory to ensure that you have matching .pyc files for each of your .py files.

A good way to test for compilation errors is to use the *zendmd* utility to import the file in question.



```
zenoss@zen241: > zendmd
Welcome to the Zenoss dmd command shell!
'dmd' is bound to the DataRoot. 'zhelp()' to get a list of commands.
>>> from ZenPacks.skills1st.bridge.modeler.plugins import BridgeDeviceMib
>>> from ZenPacks.skills1st.bridge.modeler.plugins import BridgeDeviceMib
>>>
zenoss@zen241:~> zendmd
Welcome to the Zenoss dmd command shell!
'dmd' is bound to the DataRoot. 'zhelp()' to get a list of commands.
>>> from ZenPacks.skills1st.bridge.modeler.plugins import BridgeDeviceMib
Traceback (most recent call last):
  File "<console>", line 1, in ?
  File "/usr/local/zenoss/zenoss/local/jane/ZenPacks.skills1st.bridge/ZenPacks/skills1st/bridge/m
odeler/plugins/BridgeDeviceMib.py", line 32
    )
    ^
SyntaxError: invalid syntax
>>> █
```

*Figure 99: zendmd dialogue showing successful compilation and unsuccessful compilation*

The figure above shows a successful import (in fact, two of them!) – you simply receive a command prompt back. Note that you need to specify an **object** path to the Python source file, not a **file** path. The second zendmd dialogue shows a failed compilation (I removed a closing bracket from line 32).

Note that, for some Python files you might also test compilation simply with:

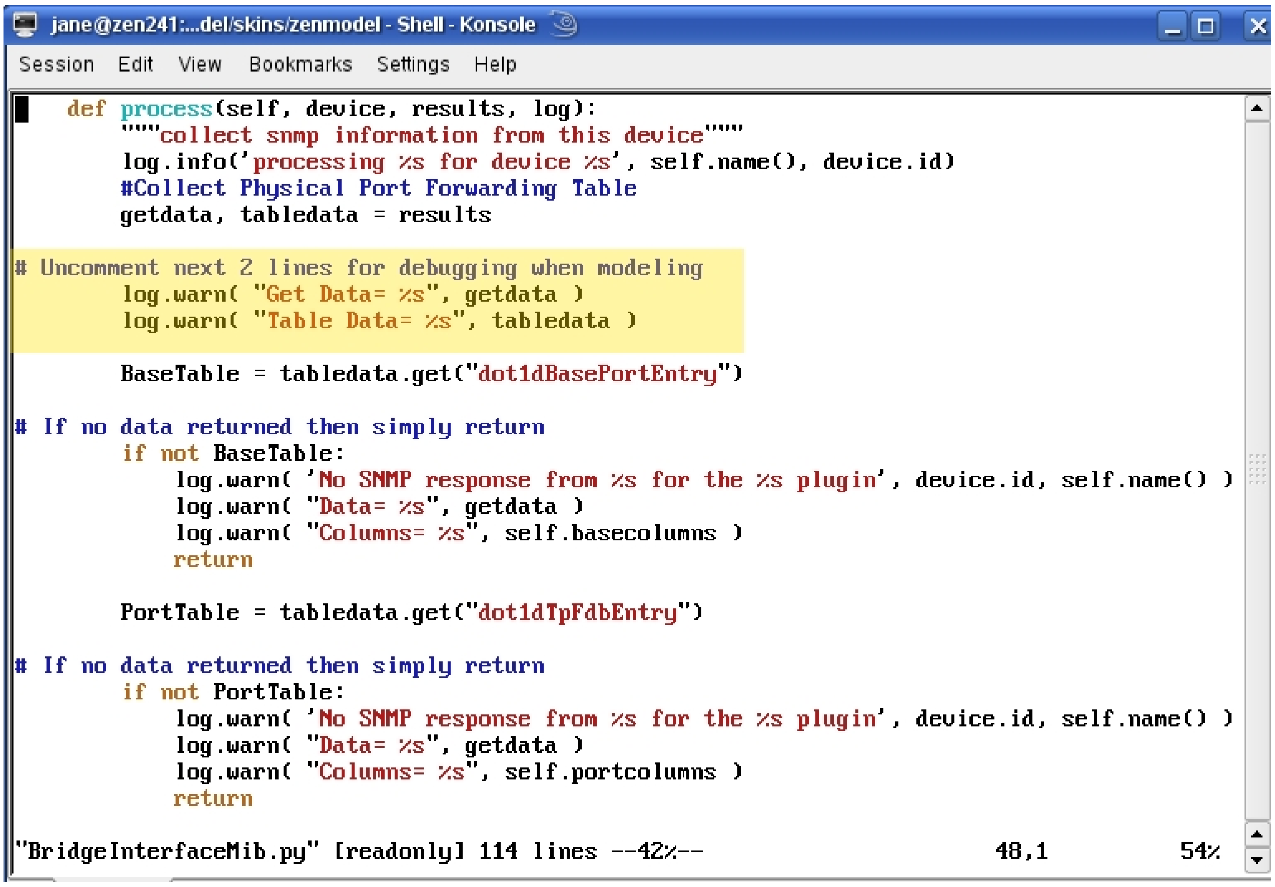
```
python BridgeMib.py
```

however, you may get different compilation errors from this test as python on its own has no concept of the Zenoss environment or libraries whereas zendmd has, and python compilation may fail with unknown imports.

If the modeler runs but fails then hopefully you get a message in the GUI showing the modeler output. If there are insufficient clues here, try running zenmodeler standalone with full debugging turned on ( *-v 10* ):

```
zenmodeler run -v10 -d group-100-s2.class.example.org
```

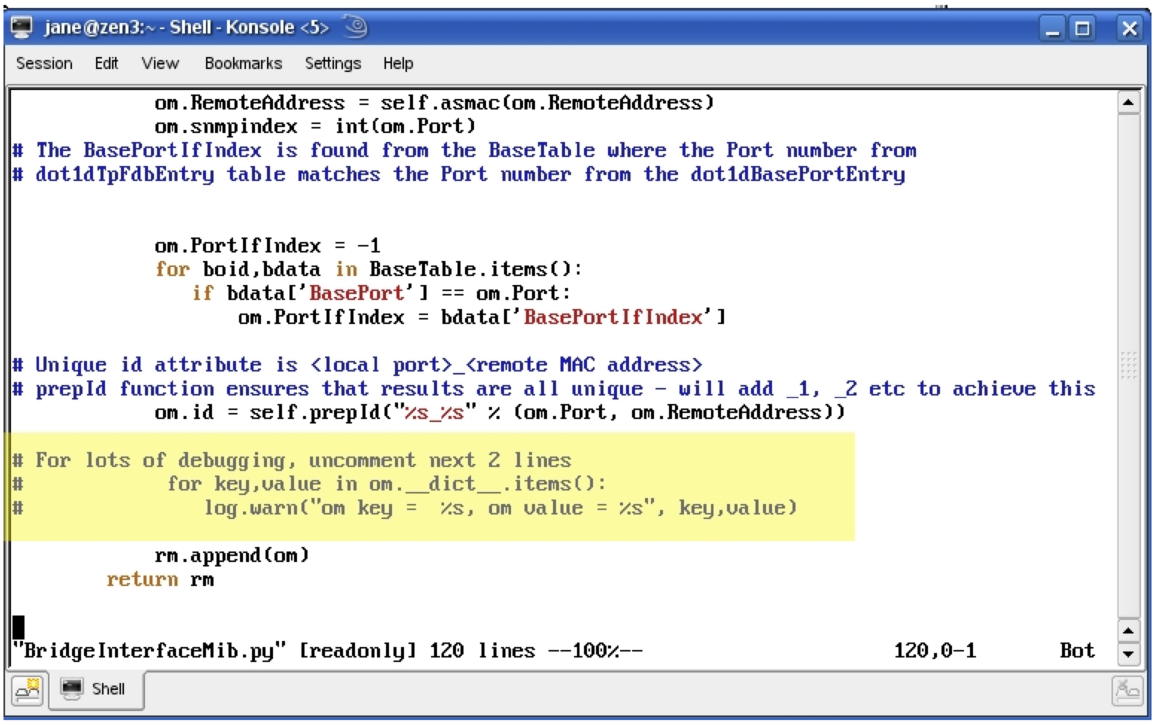
If you still can't see the problem, try putting log statements in the modeler plugin code to output intermediate data stages. Figure 100 highlights *log.warn* statements that output the results of the SNMP getdata and tabledata structures.


jane@zen241:...del/skins/zenmodel - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
    def process(self, device, results, log):
        """collect snmp information from this device"""
        log.info('processing %s for device %s', self.name(), device.id)
        #Collect Physical Port Forwarding Table
        getdata, tabledata = results

# Uncomment next 2 lines for debugging when modeling
        log.warn( "Get Data= %s", getdata )
        log.warn( "Table Data= %s", tabledata )

        BaseTable = tabledata.get("dot1dBasePortEntry")

# If no data returned then simply return
        if not BaseTable:
            log.warn( 'No SNMP response from %s for the %s plugin', device.id, self.name() )
            log.warn( "Data= %s", getdata )
            log.warn( "Columns= %s", self.basecolumns )
            return

        PortTable = tabledata.get("dot1dTpFdbEntry")

# If no data returned then simply return
        if not PortTable:
            log.warn( 'No SNMP response from %s for the %s plugin', device.id, self.name() )
            log.warn( "Data= %s", getdata )
            log.warn( "Columns= %s", self.portcolumns )
            return

"BridgeInterfaceMib.py" [readonly] 114 lines --42%--                48,1          54%
```

*Figure 100: BridgeInterfaceMib.py code highlighting debugging logging*

If you get really desperate, try the logging lines highlighted in Figure 101 to output all the attributes for an object instance; do not leave these lines uncommented once the problem is resolved.

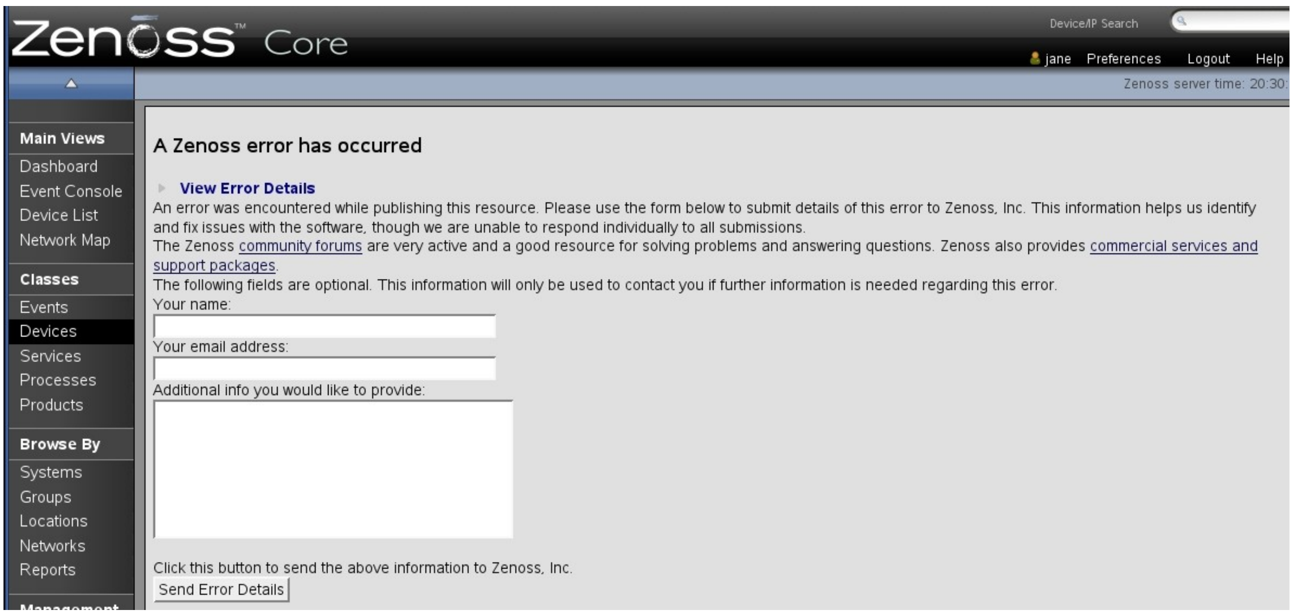

jane@zen3:~ - Shell - Konsole <5>
Session Edit View Bookmarks Settings Help
Shell

```
            om.RemoteAddress = self.asmac(om.RemoteAddress)
            om.snmpindex = int(om.Port)
# The BasePortIfIndex is found from the BaseTable where the Port number from
# dot1dTpFdbEntry table matches the Port number from the dot1dBasePortEntry


            om.PortIfIndex = -1
            for boid,bdata in BaseTable.items():
               if bdata['BasePort'] == om.Port:
                   om.PortIfIndex = bdata['BasePortIfIndex']

# Unique id attribute is <local port>_<remote MAC address>
# prepId function ensures that results are all unique - will add _1, _2 etc to achieve this
            om.id = self.prepId("%s_%s" % (om.Port, om.RemoteAddress))

# For lots of debugging, uncomment next 2 lines
#             for key,value in om.__dict__.items():
#                log.warn("om key =  %s, om value = %s", key,value)

            rm.append(om)
        return rm

"BridgeInterfaceMib.py" [readonly] 120 lines --100%--            120,0-1       Bot
```

*Figure 101: BridgeInterfaceMib .py highlighting debugging logging to output all the final object attributes*

### 6.1.3 Testing skins files and JavaScript files

If skins or JavaScript files have been created or changed, you generally only need to restart **zopectl** and then refresh the web page in the Zenoss GUI. If the code is incorrect a standard error page is shown and you can get more information by clicking the *View Error Details* link.



A Zenoss error has occurred

View Error Details

An error was encountered while publishing this resource. Please use the form below to submit details of this error to Zenoss, Inc. This information helps us identify and fix issues with the software, though we are unable to respond individually to all submissions.

The Zenoss community forums are very active and a good resource for solving problems and answering questions. Zenoss also provides commercial services and support packages.

The following fields are optional. This information will only be used to contact you if further information is needed regarding this error.

Your name:

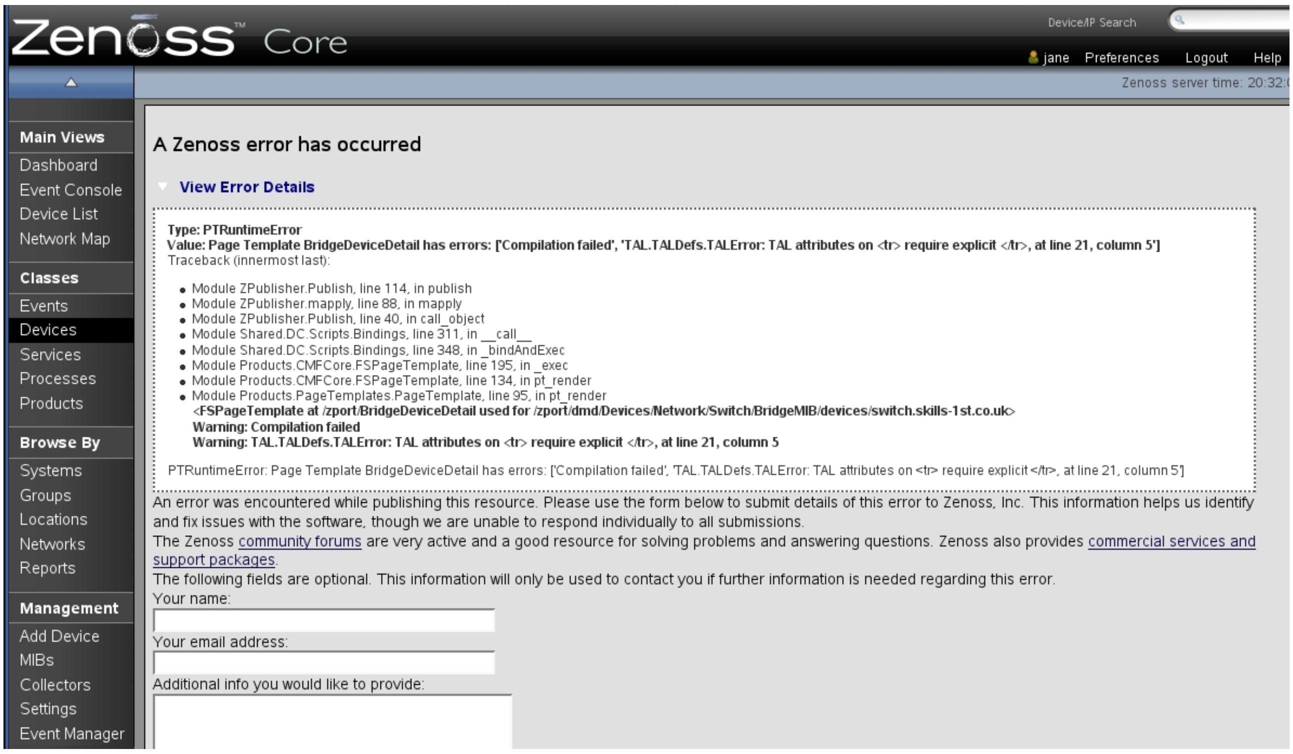
Your email address:

Additional info you would like to provide:

Click this button to send the above information to Zenoss, Inc.

Send Error Details

*Figure 102: Standard error message for a faulty web page definition*



A Zenoss error has occurred

**View Error Details**

```
Type: PTRuntimeError
Value: Page Template BridgeDeviceDetail has errors: ['Compilation failed', 'TAL.TALDefs.TALError: TAL attributes on <tr> require explicit </tr>, at line 21, column 5']
Traceback (innermost last):

  • Module ZPublisher.Publish, line 114, in publish
  • Module ZPublisher.mapply, line 88, in mapply
  • Module ZPublisher.Publish, line 40, in call_object
  • Module Shared.DC.Scripts.Bindings, line 311, in __call__
  • Module Shared.DC.Scripts.Bindings, line 348, in _bindAndExec
  • Module Products.CMFCore.FSPageTemplate, line 195, in _exec
  • Module Products.CMFCore.FSPageTemplate, line 134, in pt_render
  • Module Products.PageTemplates.PageTemplate, line 95, in pt_render
    <FSPageTemplate at /zport/BridgeDeviceDetail used for /zport/dmd/Devices/Network/Switch/BridgeMIB/devices/switch.skills-1st.co.uk>
    Warning: Compilation failed
    Warning: TAL.TALDefs.TALError: TAL attributes on <tr> require explicit </tr>, at line 21, column 5

PTRuntimeError: Page Template BridgeDeviceDetail has errors: ['Compilation failed', 'TAL.TALDefs.TALError: TAL attributes on <tr> require explicit </tr>, at line 21, column 5']
```

An error was encountered while publishing this resource. Please use the form below to submit details of this error to Zenoss, Inc. This information helps us identify and fix issues with the software, though we are unable to respond individually to all submissions.
The Zenoss community forums are very active and a good resource for solving problems and answering questions. Zenoss also provides commercial services and support packages.
The following fields are optional. This information will only be used to contact you if further information is needed regarding this error.
Your name:
Your email address:
Additional info you would like to provide:

*Figure 103: View Error Details for a faulty web page definition*

Figure 103 shows the detailed error output. The file and line number at fault are documented (I had indeed commented out a closing </tr> at line 21 of the BridgeDeviceDetail.pt file). Simply fix the file, issue *zopectl restart* and refresh the web page.

Sometimes the web View Error Details page suggests something is wrong that is nowhere near anything you have recently changed. If this happens, try restarting the whole Zenoss system with:

```
zenoss stop
zenoss start
```

If you suspect issues with JavaScript you could run a Firefox Error Console to see the JavaScript errors - “There will be tons of CSS issues coming from different CSS pages (it's annoying, but not fatal), and you can safely ignore them”, says the debugging section 13.8 of the Zenoss Developer's Guide! If you filter out Warning severity messages from the Firefox Error Console, it may help you spot real issues.

Section 13.8 also has the following advice with regard to JavaScript:

“The Firefox Error Console will not tell you if Firefox wasn't able to find or load a JavaScript file (if the path you've specified in your Web page to get to the JavaScript file is incorrect). In order to determine if Zope was given a path to a filename that it couldn't find, you'll need to go into Zope's ZMI, go to the error log (http://yourzenossserver:8080/error_log/manage) and remove all of the error log filters.

After you do that, retry the operation and you can see what files Zope wasn't able to find and fix the paths in your page."

### 6.1.4 Debugging problems with performance data

There are several ways that performance data collection can fail:

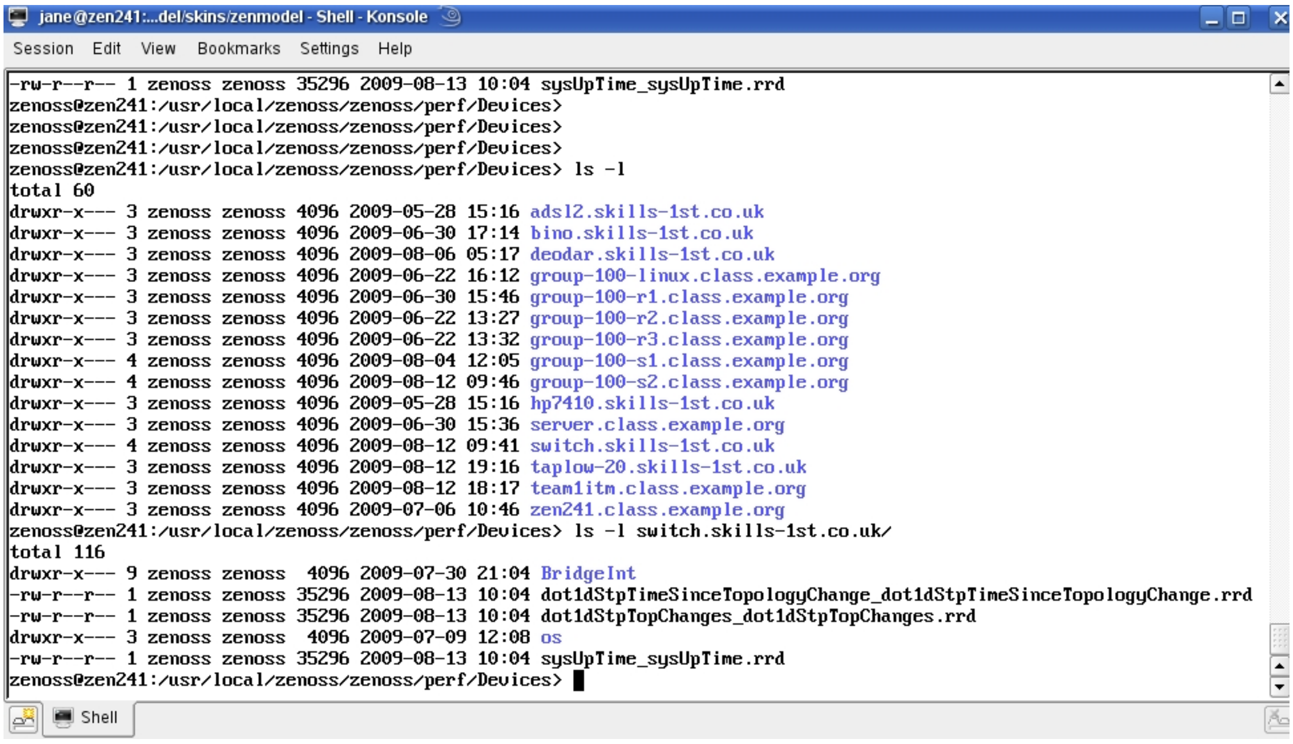
- A template is created but not **bound** to a device. In this case, no attempt will be made to collect data. Go to the device's details page and check the *Monitoring Templates* listed there (remember that component templates, matching the component object class name, do **not** need binding – this happens automatically – and the component template will not be listed for the device).
- You could also check the *zDeviceTemplates* property from the *Configuration Properties* menu to ensure the correct templates are bound.
- Scalar MIB values need the trailing *.0*; otherwise no data will be collected.
- If SNMP community names configured in Zenoss do not match those in the target agents then you will get no SNMP data. Test with a simple snmpwalk command from a command line; for example:

  ```
  snmpwalk -v 1 -c public switch.skills-1st.co.uk system
  ```
- If a template is correctly configured and bound but there are only one or two data values collected (counter values need at least two values before a point can be plotted as it is a rate-of-change measurement), you will see a graph with no data and the *cur, avg* and *max* values will have the value **nan**. This simply means graph points are not yet available; another snmp polling interval usually fixes this issue.
- For component device templates collecting tables of SNMP data, the instance may be the issue. Increasing the logging level for zenperfsnmp may help diagnose this.

Templates collect data into Round Robin Database (rrd) files, held under $ZENHOME/perf/Devices with a separate subdirectory for each device and each device may have subdirectories for components such as *os* or *BridgeInt* (ie. the **relationship** name of the contained device); there may be further subdirectories for each instance (ie each port for the BridgeMIB ZenPack), where the subdirectory name is the id of the port, for example, *24_00_30_E2_F7_2A_1C*.

Always check that rrd files exist. Templates for devices have the format:

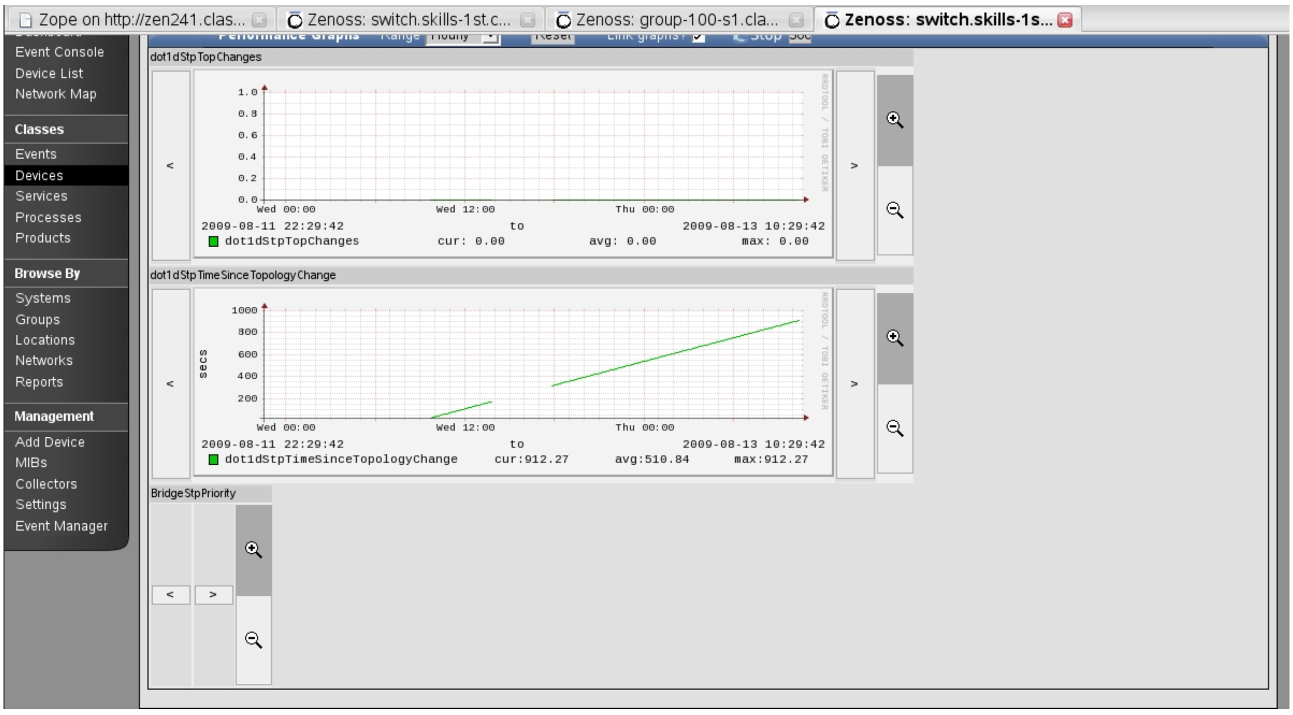
```
<datasource name>_<datapoint name>.rrd
```


jane@zen241:...del/skins/zenmodel - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
-rw-r--r-- 1 zenoss zenoss 35296 2009-08-13 10:04 sysUpTime_sysUpTime.rrd
zenoss@zen241:/usr/local/zenoss/zenoss/perf/Devices>
zenoss@zen241:/usr/local/zenoss/zenoss/perf/Devices>
zenoss@zen241:/usr/local/zenoss/zenoss/perf/Devices>
zenoss@zen241:/usr/local/zenoss/zenoss/perf/Devices> ls -l
total 60
drwxr-x--- 3 zenoss zenoss 4096 2009-05-28 15:16 adsl2.skills-1st.co.uk
drwxr-x--- 3 zenoss zenoss 4096 2009-06-30 17:14 bino.skills-1st.co.uk
drwxr-x--- 3 zenoss zenoss 4096 2009-08-06 05:17 deodar.skills-1st.co.uk
drwxr-x--- 3 zenoss zenoss 4096 2009-06-22 16:12 group-100-linux.class.example.org
drwxr-x--- 3 zenoss zenoss 4096 2009-06-30 15:46 group-100-r1.class.example.org
drwxr-x--- 3 zenoss zenoss 4096 2009-06-22 13:27 group-100-r2.class.example.org
drwxr-x--- 3 zenoss zenoss 4096 2009-06-22 13:32 group-100-r3.class.example.org
drwxr-x--- 4 zenoss zenoss 4096 2009-08-04 12:05 group-100-s1.class.example.org
drwxr-x--- 4 zenoss zenoss 4096 2009-08-12 09:46 group-100-s2.class.example.org
drwxr-x--- 3 zenoss zenoss 4096 2009-05-28 15:16 hp7410.skills-1st.co.uk
drwxr-x--- 3 zenoss zenoss 4096 2009-06-30 15:36 server.class.example.org
drwxr-x--- 4 zenoss zenoss 4096 2009-08-12 09:41 switch.skills-1st.co.uk
drwxr-x--- 3 zenoss zenoss 4096 2009-08-12 19:16 taplow-20.skills-1st.co.uk
drwxr-x--- 3 zenoss zenoss 4096 2009-08-12 18:17 team1itm.class.example.org
drwxr-x--- 3 zenoss zenoss 4096 2009-07-06 10:46 zen241.class.example.org
zenoss@zen241:/usr/local/zenoss/zenoss/perf/Devices> ls -l switch.skills-1st.co.uk/
total 116
drwxr-x--- 9 zenoss zenoss  4096 2009-07-30 21:04 BridgeInt
-rw-r--r-- 1 zenoss zenoss 35296 2009-08-13 10:04 dot1dStpTimeSinceTopologyChange_dot1dStpTimeSinceTopologyChange.rrd
-rw-r--r-- 1 zenoss zenoss 35296 2009-08-13 10:04 dot1dStpTopChanges_dot1dStpTopChanges.rrd
drwxr-x--- 3 zenoss zenoss  4096 2009-07-09 12:08 os
-rw-r--r-- 1 zenoss zenoss 35296 2009-08-13 10:04 sysUpTime_sysUpTime.rrd
zenoss@zen241:/usr/local/zenoss/zenoss/perf/Devices> 
```

*Figure 104: Directories for performance files for devices*

If you see graphs that have no data at all, this generally means that a template is bound but there is no rrd file, as shown in Figure 105.

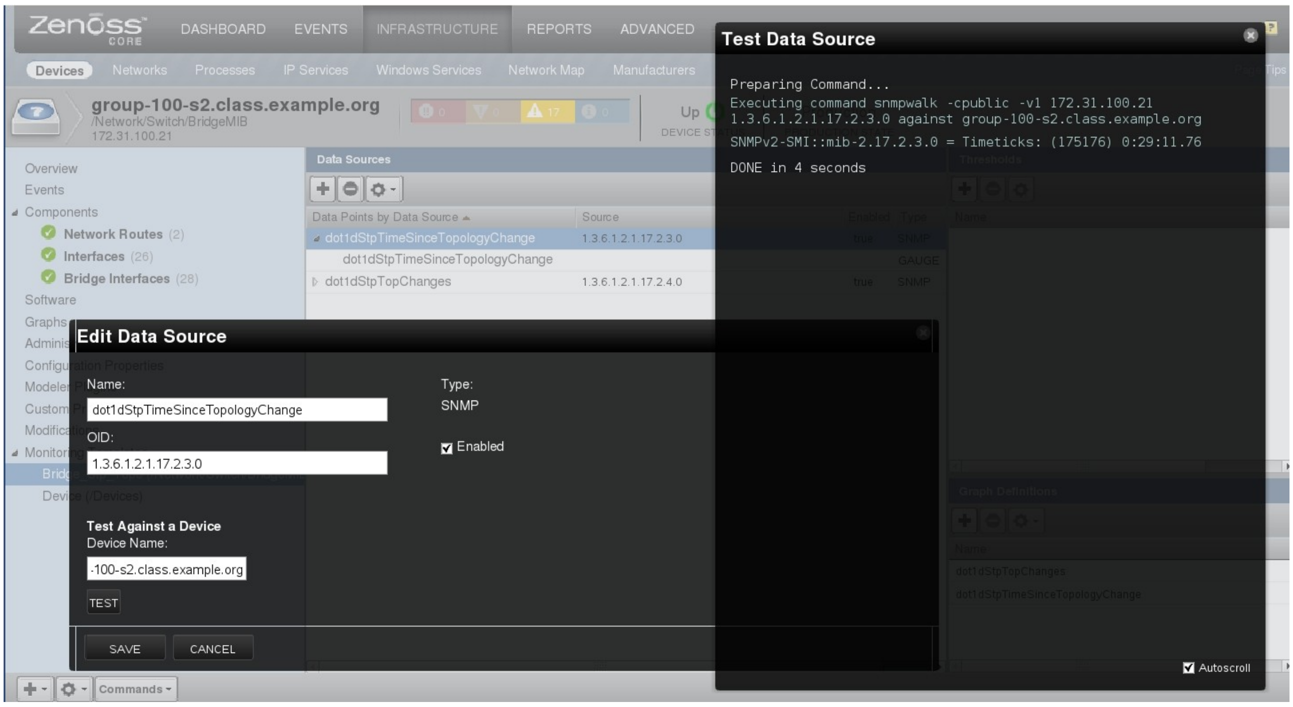


```
2009-08-11 22:29:42            to            2009-08-13 10:29:42
■ dot1dStpTopChanges        cur: 0.00     avg: 0.00     max: 0.00
```

```
2009-08-11 22:29:42            to            2009-08-13 10:29:42
■ dot1dStpTimeSinceTopologyChange    cur:912.27   avg:510.84   max:912.27
```

*Figure 105: The empty graph at the bottom suggests that a template is bound but no data has been collected*

Note that when you configure data sources in a template, there is a test button that you can use to specify a device known to Zenoss; however, the test that is run, strictly, is an *snmpwalk* whereas the zenperfsnmp daemon is more likely to issue an *snmpget,* so the test button can disguise problems with instances.



group-100-s2.class.example.org
/Network/Switch/BridgeMIB
172.31.100.21

| Data Points by Data Source | Source | Enabled | Type |
|---|---|---|---|
| dot1dStpTimeSinceTopologyChange | 1.3.6.1.2.1.17.2.3.0 | true | SNMP |
| dot1dStpTimeSinceTopologyChange | | | GAUGE |
| dot1dStpTopChanges | 1.3.6.1.2.1.17.2.4.0 | true | SNMP |

Edit Data Source

Name: dot1dStpTimeSinceTopologyChange

OID: 1.3.6.1.2.1.17.2.3.0

Type: SNMP

✓ Enabled

Test Against a Device
Device Name: -100-s2.class.example.org

Test Data Source

```
Preparing Command...
Executing command snmpwalk -cpublic -v1 172.31.100.21
1.3.6.1.2.1.17.2.3.0 against group-100-s2.class.example.org
SNMPv2-SMI::mib-2.17.2.3.0 = Timeticks: (175176) 0:29:11.76
DONE in 4 seconds
```

*Figure 106: Using the TEST button from the Edit Data Source configuration dialogue*

### 6.1.5 General testing and debugging hints and tips

A common issue with some environments and browsers is to see a blank screen in the Zenoss GUI. This is usually resolved simply by resizing fonts in the browser using Ctrl - .

There are two general areas for debugging help. Zenoss logfiles are all held under *$ZENHOME/log*. By default they have an *Info* level of logging but this can be increased to *Debug* to provide lots more data. When the problem is resolved, the original logging level should be restored.

Daemon log files and their configuration can be inspected from the *ADVANCED -> Settings -> Daemons* menu. To increase the debug level, change the *logseverity* to *Debug*. If you check the configuration file for this daemon in *$ZENHOME/etc* you will see a line:

```
    logseverity   10
```

Any changes to a daemon's configuration file requires a restart of the daemon, either through the GUI or using *<daemon> restart* from a command line.

In addition to checking specific Zenoss daemon files like zenmodeler.log or zenperfsnmp.log, it is always worth also checking **zenhub.log** and **event.log.**

The second general debugging tool is **zendmd**. This is a Python interpretive environment provided by Zenoss that already understands some of the Zenoss object hierarchy. It is an excellent "sandpit" to test out bits of Python and to query Zenoss objects and their attributes and methods. Several examples have already been demonstrated throughout this document.

When weird things happen that really make no sense at all, try recycling the whole Zenoss system with a:

```
zenoss stop
zenoss start
```

## 7 Conclusions

ZenPacks are a powerful and flexible way to extend core Zenoss capability. Development mode provides a simple method to achieve simple ZenPacks. Source mode ZenPacks require more understanding of Zenoss internals, SNMP, Python and of JavaScript, but anything is possible.

The Bridge MIB ZenPack will be available from the Zenoss ZenPacks website - http://www.zenoss.com/community/projects/zenpacks/ . The source code for the ZenPack is available, with this document, at http://www.skills-1st.co.uk/papers/jane/zenpacks/ .

## References

1. Zenoss Developer's Guide 3 - http://community.zenoss.org/community/documentation
2. Zenoss Administration Guide 3 - http://community.zenoss.org/community/documentation
3. Zenoss Extended Monitoring Guide 3 for documentation on Core and Enterprise ZenPacks - http://community.zenoss.org/community/documentation
4. Zenoss FAQ at http://community.zenoss.org/docs/DOC-2446 and http://community.zenoss.org/docs/DOC-4724
5. Zenoss ZenPacks site at http://community.zenoss.org/community/zenpacks
6. Zenoss-ZenPacks forum at http://community.zenoss.org/community/forums/zenoss-zenpacks
7. Zenoss community developer site wiki at http://community.zenoss.org/docs/DOC-2350 , "Diving into the device model".
8. "Custom ZenPacks rough guide" contributed by ***blacks*** to the Zenoss forum at http://community.zenoss.org/docs/DOC-2358
9. Zenoss download site - http://community.zenoss.org/community/download
10. net-SNMP SNMP agent from http://www.net-snmp.org/
11. BRIDGE MIB, RFC 1493 - http://www.ietf.org/rfc/rfc1493.txt
12. oidview online website for viewing MIBs such as the BRIDGE MIB - http://www.oidview.com/mibs/0/BRIDGE-MIB.html
13. "Learning Python" by Mark Lutz, published by O'Reilly
14. Zope web application server information from http://www.zope.org/WhatIsZope
15. "The Zope2 Book" from http://docs.zope.org/zope2/zope2book/
16. Zope Page Templates Reference - http://docs.zope.org/zope2/zope2book/AppendixC.html
17. Zope Configuration Markup Language (ZCML) reference - http://apidoc.zope.org/++apidoc++/ - and follow the ZCML link
18. Zope 3 Interfaces reference - http://wiki.zope.org/zope3/WhatAreInterfaces
19. "ZenPack Development Procedures" document for working with ZenPacks on Zenoss's ZenPack site, written by David Buler ("phonegi") - http://community.zenoss.org/docs/DOC-10223

## Acknowledgements

Several people have contributed either actively or passively to this paper:

- "blacks" on the Zenoss forum for his Custom ZenPack Rough Guide that got me started. The original work for this was submitted by Zach Davis.
- Danny Deng who sent me his ZenPack samples and explanations
- George Fakhri for his blog post on "How to create a ZenPack.."
- "bigegor" on the Zenoss forum for his excellent ZenPacks used extensively as examples, and for his responses to questions
- David Buler ("phonegi") contributed hugely by doing the "detective work" on the mechanics of the new component panel code