opCharts Resource Model Description

Introduction

Resources help present a normalised view of NMIS models. For example they allow for taking different SNMP implementations of CPU's (avgBusy5 /ssCpuRawUser/etc) into a single normalised resource with variables named the same so systems like TopN can look at all CPU's and present a list of them. In order to do this the Resource model definitions are defined per-model, they can import common sections, and override any part of them. Each section defines how to find the information in the NMIS model, and how it will be presented.

Internal (dev) Info

Internally, resources are accessed like this:
Node -> Resource Types -> Resource Keys -> Resource

Resources are all indexed and are accessed via their key, if there is only one of them then there is only 1 key available. Resource types can be asked for the available keys, then, can be asked for a resource using one of the keys provided.

end of internal dev info.

Example Resource Types:

- cpu (display name CPU)
- memory (display name Memory)
- interface (display name Interface)
- packets (display name Packets)
- processes (display name Processes)
- users (display name Users)

New resource types can be created by simply adding a new section to a resource file, when naming keep in mind that these are meant to be common sections shared over all models (but don't have to be).

Resource models are used in opCharts TopN analysis, opTrend TopN as well as the opCharts Node panel (when enabled).

Resource Definition

Resource section layout example:

Note: the resource MUST contain a property named "resource_enabled", which tells the system if this resource is enabled or disabled. If this already exists as a property of the thing being resource-ified, that will do, if not, create a virtual property which defines it.

```
resources => {
                # this key defines the resource type
                'cpu' => {
                        # the name that is displayed for this resource
                        'resource_name' => 'CPU',

                        # resources can inherit, and then override, use import to specify the file/section to
import from
                        'import' => 'Resources-Common/cpu',

                        # is the NMIS model section we point at indexed?
                        'indexed' => 'true',

                        # which NMIS model section to look in (other egs: 'system','interface','storage', etc.
these are on the base level)
                        # also points into the -node file, tells it which section in the node file to use
                        'section' => 'device',

                        # list of properties which are appended to create the index for this, they are joined
together with ":"
                        # and used as the resource name
                        # not required if indexed is false
                        'indexed_name' => ['index','hrDeviceDescr'],
```

```
                        # graph_type, this is passed into getDBName to resolve the db path
                        'rrd_resource' => 'hrsmpcpu',

                        # array of graphs that relate to this resource
                        'graphs' => ['hrsmpcpu'],

                        # not listed: properties
                        # properties are gleaned from the -node file, anything available in the section that
relates to this
                        # resource are available

                        # list of virtual properties, see "virtual options", can access any properties,virtual
properties
                        'virtual_properties' => [
                                {
                                        name => 'resource_enabled',
                                        operation => 'define',
                                        value => '1',
                                },
                                {
                                        name => 'friendly_name',
                                        operation => 'calculation',
                                        value => '"CPU $index"',
                                }
                        ],

                        # list of fields available from the RRD, fields must be listed here to be used, even if
they surely exist in the rrd
                        'fields' => [
                                {
                                        name =>'hrCpuLoad',
                                        display_suffix => '%'
                                }
                         ],

                        # list of virtual fields, see "virtual options", can access any properties,virtual
properties,fields or other virtual fields
                        # NOTE: be sure not to create circular dependencies (x uses y for calculation, y uses x
for calculation)
                        'virtual_fields' => [
                                {
                                        name => 'cpuLoad',
                                        operation => 'define',
                                        value => '$hrCpuLoad',
                                        display_suffix => '%'
                                }
                        ],

                        # this is for opTrend, can be ignored
                        # tells optrend which fields to run seds on (virtual or normal)
                        'seds_fields' => ['cpuLoad'],

                        # also for opTrend, can be ignored
                        # tells optrend which fields to run thresholding on, and which type to use
                        'threshold_fields' => {
                                'cpuLoad' => 'ev',
                        }
                }
        }
```

Virtual Options

Virtual options are run through a parser, they are resolved on access so none of the data is actually parsed/calculated until the data is required.

```
{
        name => "name of the thing"
        operation => type, explained below
        value => depends on type
        display_suffix => when shown on screen, what suffix to use, eg bytes, %, whoosits/whenzees
}
```

There are 3 operation types of for virtual field/properties:

1. define - create this property/field, give it the value specified. Handy for renaming fields
2. define_if_not_defined - create this property and set this value, but only if it does not already exist. This functionality could be expressed in a calculation but this is faster
3. calculation - calculate the value of this property/field, variables include any property/field, prepended with $. Properties can not access field data, fields can access properties. Any perl can be put in here, just needs to return a value (and not have circular references).

Not really renaming, but defining a second virtual field that has the same value (but different name) as an existing field. Renaming field example:

```
'fields' => [
        {
                name => 'hrProcesses',
                display_suffix => ' processes'
        }
],
'virtual_fields' => [
        {
                name => 'hrSystemProcesses',
                operation => 'define',
                value => '$hrProcesses',
        },
]
```