Advanced Modelling: When a single SNMP variable isn't enough

Occasionally you will come across a device or a situation where collecting a single SNMP variable is insufficient, for example when two or more SNMP properties need to be combined to provide a meaningful measurement.

NMIS version 8.4.8G and later support modelling such scenarios using custom variables, or `CVARs`. With this mechanism you can temporarily capture up to 10 separate SNMP properties as a `CVAR` and define an arbitrarily complex expression (in perl) that transforms these `CVARs` into the one measurement that you want to collect and/or display.

Where and How to use CVARs

`CVARs` are supported

- in the `test` and `value` expressions in the NMIS alert and threshold subsystem,
- in `calculate` expressions in the general modelling subsystem,
- and from NMIS version 8.6 on, also in `control` expressions evereywhere (in versions before that only a single `CVAR` was supported in `control`)

To use `CVARs` you define the required `CVARs` as holding a previously specified SNMP variable at the beginning of one of the supported expressions; Subsequently you can then reference the `CVAR` value in the part of the expression that calculates the desired value to be used by NMIS.

An example scenario

The DS3 MIB defines a variety or error counters for DS3 circuits like "dsx3CurrentLCVs" which are based on a 15 minute observation interval and reset automatically at the end of the interval. As the interval start and end is arbitrary and up to the device to set, just capturing the error counters themselves is not quite workable. However, the DS3 MIB also specifies the variable "dsx3TimeElapsed" that holds the seconds elapsed since the start of the current observation interval. Dividing the raw error counter by the number of seconds into the interval results in a normalised errors-per-second rate which works well for collection and display.

Here is an excerpt of the relevant model file:

```
'systemHealth' =>
{
        'sections' => 'ds3Errors',
        'sys' =>
        {
                'ds3Errors' =>
                {
                        'indexed' => 'dsx3CurrentIndex',
                        'index_oid' => '1.3.6.1.2.1.10.30.6.1.1',
                        'headers' => 'ds3intf,ds3linestatus',
                        'snmp' => {
                                'ds3intf' => {
                                        'oid' => '1.3.6.1.2.1.2.2.1.2', # ifDescr
                                        'title' => 'DS3 Interface',
                                },
                                'ds3linestatus' => {
                                        'oid' => '1.3.6.1.2.1.10.30.5.1.10', # dsx3LineStatus
                                        'title' => "DS3 Line Status",
                                        'calculate' => 'my @x; my %triggers=(1,"No Alarm",2,"Rx Remote Alarm",
4,"Tx Remote Alarm",8,"Rx AIS",16,"Tx AIS",32,"Rx LOF",64,"Rx LOS",128,"Loopback",256,"Test Pattern",512,"
Unknown",1024,"Near end unavailable signal",2048,"Carrier Equip OOS"); while (my ($num,$txt)=each(%triggers)) {
push (@x,$txt) if (int($r) & int($num)); }; return join(", ",@x); ',
                                },
...
                                'ds3LCV' => {
                                        'oid' => '1.3.6.1.2.1.10.30.6.1.6', # dsx3CurrentLCVs
                                        'title' => 'Line Coding Violations per second',
                                        'calculate' => 'CVAR1=ds3Elapsed; return ($CVAR1? $r/$CVAR1 : 0);',
                }
        },        # sys

        'rrd' => {
                'ds3Errors' => {
                        'indexed' => 'true',
                        "graphtype" => "ds3Errors",
                        "snmp" => {
                                'ds3Elapsed' => {
                                        'oid' => '1.3.6.1.2.1.10.30.5.1.3', # dsx3TimeElapsed
                                        'title' => 'elapsed seconds in current measurement interval',
                                        'option' => 'gauge,0:U',
                                },
...
                                "ds3LCV" => {
                                        'oid' => '1.3.6.1.2.1.10.30.6.1.6',
                                        'option' => 'gauge,0:U',
                                        'title' => "Line Coding Violations per second",
                                        'calculate' => 'CVAR1=ds3Elapsed; return ($CVAR1? $r/$CVAR1 : 0);',
                                },
        },         # rrd
},        # systemhealth
```

In the example above, the `calculate` expressions are used in two ways:

- to transform the bitfield variable "DS3 Line Status" into a more verbose textual list of component statuses,
- and to divide the raw `dsx3CurrentLCVs` error count by the `dsx3TimeElapsed` interval length.

In both cases the syntax is very straight-forward:

- The expression must be a valid perl statement and return exactly one value.
- The tokens `$r`, and `CVAR0` to `CVAR9` are interpreted by NMIS; everything else is perl.
- Defining and using local variables with `my` is ok, but don't attempt to change any global NMIS variables.
- "`CVAR1=some_snmp_var;`" defines what SNMP object CVAR1 is supposed to hold. The parser understands `CVAR0` to `CVAR9` for a total of 10 captures.
- You can use functions that were defined elsewhere in NMIS in your `calculate` expression.
  You will likely have to include the full module namespace in the function call, e.g. `func::beautify_physaddress(...)`.
  Only functions without side-effects should be used.
- "`return $r/$CVAR1;`" accesses the value of `CVAR1` in an expression. The variable "`$r`" represents the SNMP variable that the `calculate` expression is attached to.

Please note that

- the `$CVARn` replacement in the expression is performed on a purely textual basis, before the expression is handed to the perl interpreter for evaluation :
    - For string variables you have to provide quotes in your expression, e.g.

      ```
      calculate => 'CVAR1=somestringthing; return 42 if ("$CVAR1" eq "online");'
      ```

    - Numeric variables can be used straight without quotes.
- the `$CVARn` access refers to the *raw* value of the named property, ie. the data before any `replace` or `calculate` expressions for the named property were evaluated.

How to keep temporary CVAR data out of the RRD databases

As outlined above all the objects that you want to access via `CVARs` must be defined in the same section. If your test/calculate expression is within an `rrd` section, all the other objects will have to be within that `rrd` section, too, and thus they would be collected by NMIS and stored in RRD - quite wasteful if these other variables are just temporary and only there to for access using one `CVAR` expression.

In versions 8.6.0 and above you can prevent this by adding an `option` with value `nosave`:

```
'snmp' => {
  'hrNumUsers' => {
     'oid' => 'hrSystemNumUsers',
     'option' => 'nosave',
  },
```

In the example above, `hrNumUsers` would be retrieved with SNMP, and other variables could be defined in terms of e.g. `CVAR3=hrNumUsers`, but `hrNumUsers` would not be saved.

Please note that setting `nosave` disables alerts for the given object.

Using Placeholders

Sometimes there is a need to create calculated data using data gathered from several sources and calculated into a single table. Situations such as this might call for use of a plugin (NMIS 9 Collect and Update Plugins). The problem comes in when the model tries to interpret the sections in the model when there is lack of data to do so. This causes errors in the model. To solve this, there is a keyword. 'placeholder' which can be used in the 'sys' section of the model definition to tell the model that the 'headers', labels, and 'titles' only exist to create the necessary infrastructure for the data, and the some other method will be used to create the values. There really isn't much to this, just the keyword 'placeholder', and the value will be printed in the log to say that the <value> will be used to fill in the data as shown in the example below.

```
      'ciscoNormalizedCPUMem' => {
        'headers' => 'TotalCPUs,MemoryUsedMax,MemoryUsed,MemoryFreeMax,MemoryFree',
        'placeholder' => 'plugin',
        'graphtype' => 'health',
        'indexed' => 'true',
        'snmp' => {
          'TotalCPUs' => {
            'title' => 'Number of CPUs'
          },
          'MemoryUsedMax' => {
            'title' => 'Maximum Memory Used'
          },
          'MemoryUsed' => {
            'title' => 'Current Memory Used'
          },
          'MemoryFreeMax' => {
            'title' => 'Maximum Memory Free'
          },
          'MemoryFree' => {
            'title' => 'Current Memory Free'
          },
        },
      },
```