Custom Tables in NMIS

Introduction

While working with customers who wanted to extend NMIS and make it even more of a Network Management System (e.g. to support parts of their operational processes and integrate more closely with their ITIL service management processes), we found that it could become difficult for them to maintain the customisations of the extended data collection. To better support NMIS users we have simplified the way "Tables" in NMIS are defined and extended as well as how they are shown in the Menus.

This article will briefly describe how this capability works and how it supports operational agility.

Prerequisites

- NMIS version 8.3.18G or newer for custom tables
- NMIS version 8.6.2G or newer for input validation
- Shell access to the NMIS server and suitable user privileges to edit the NMIS configuration files (which usually means being a member of the group "nmis" or having root privileges)

Custom Tables in NMIS

What are NMIS Tables?

To use NMIS various data is required, this data represents various policies, configuration, credentials or a combination of all of those. In the past NMIS users have added tables as they needed, this required some Perl coding. To support faster and more easily modified tables in NMIS the table definitions are now defined outside of the code base, making the tables themselves configuration items. So like the chicken and the egg, you need to start with something.

Typical Tables used in NMIS

The following tables are used in NMIS internally:
(Note that this also includes the 'meta-table' `Tables.nmis`, which defines what other tables NMIS presents, to allow dynamic definition of tables.)

File	Description
Access.nmis	Access levels for Authorisation System
BusinessServices.nmis	A list of Business Services to link to a node.
Contacts.nmis	Contacts information used for notifications.
Enterprise.nmis	List of “vendors” SNMP OID prefixes
Escalations.nmis	Escalation policy, how notifications will happen
Links.nmis	List of Links in the network.
Locations.nmis	List of Locations
Logs.nmis	Log viewer configuration file

Modules.nmis	Opmantek modules integration
Nodes.nmis	Main NMIS8 Nodes file
Outage.nmis	Current planned outages (deprecated in 8.6.2G)
Portal.nmis	Portal configuration for internal integrations
PrivMap.nmis	Privilege mappings for authorisation
Services.nmis	Services configuration file
ServiceStatus.nmis	The definition of the Service Status's for NMIS (production, pre-production, etc)
Toolset.nmis	External tools configuration file
Tables.nmis	The list of Tables in NMIS
Users.nmis	Users authorisation mappings
ifTypes.nmis	List of standard interface types from IANA
Config.nmis	8.6.2G and newer: the main configuration file is now treated as a table, with validation and customisation features like the others.

Table Configuration

For each known type of table there is a *separate* "table configuration" file, all of which are named `Table-<yourtable>.nmis` (e.g. for table `Users.nmis` the configuration file is called `Table-Users.nmis`).
Both the actual tables and their configuration files live in the `conf` directory, ie. `/usr/local/nmis8/conf`.

These table configuration files consist of declaration stanzas and little bits of code, which is evaluated at run time.

Let's look at an example; this is what "Table-SampleTable.nmis" could look like:

```
%hash = (
  SampleTable => [
    { Email => { header => 'Email Address', display => 'key,header,text', value => [""] }},
    { Name => { header => 'Name', display => 'header,text', value => [""] }},
    { Age => { header => 'Age', display => 'header,text', value => [""],
               validate => { "int" => [ 0, undef ] }}},
    { Married => { header => 'Married', display => 'popup', value => ["true", "false"] }},
    { Children => { header => 'Children', display => 'popup', value => [0,1,2,3,4,5,6,7,8,9,10,11,12,"Many"] }},
  ]
);
```

`SampleTable => [`	Is the name of the table, this should match the name, e.g. Table-**SampleTable**.nmis

`{`
`  Email => {`
`    header => 'Email Address',`
`    display => 'key, header,text',`
`    value => [""]`
`  }`
`},`
	Each Column in the table is defined with an entry like this. In this case the column is called **Email**
	
	To define each column necessary fields are:
	
	○ header - is the what will be displayed when the table is viewed.
	○ display - header indicates if it should be in the header or not, and text indicates what sort of input box to use. T his includes the work key if it is to be included as the primary key.
	○ value - what is the default value or select list.

{
 	
  Married => {
 	
  header =>
'Married',
 	
  display =>
'popup',
 	
  value =>
["true", "false"]
 	
  }
 	
},
 	
	This field would not be displayed as a textbox in the main view but instead would contain a select list (drop down) to select true or false from.

Display Options

Display controls how the field from the table configuration will be displayed and where it will be displayed, it is a comma separated list of values as shown in the examples above.

Possible values for display are:

Value	Description	NMIS Version
header	If the value "header" is present, the field will be displayed when viewing the table. All fields are visible when editing an entry.	NMIS 8.1.1
key	This value is to be used as a key value, if multiple key values are defined, they will be combined together to make the key of the record. It is recommended to use a single value for a key value.	NMIS 8.1.1
readonly	A readonly field will not be editable, which means it must be added automatically as in the case of something like a UUID or edited from Unix.	NMIS 8.3.19 G
text	The standard field is a "text" field, this is equivalent to a HTML "input" form element.	NMIS 8.1.1
textbox	A text box being a little larger higher, this is equivalent to a HTML "textarea" form element.	NMIS 8.3.19 G
popup	A single value select box, this is equivalent to a HTML "select" form element.	NMIS 8.1.1
scrolling	A multiple select box, where you can select one or more values. This is equivalent to a HTML "select" form element with the attribute of "multiple" set to "multiple".	NMIS 8.1.1

Validation Options (8.6.2G and newer)

From NMIS version 8.6.2G onwards, a basic data validation mechanism is available for all NMIS tables. Not all properties are setup for validation yet, but the most critical ones are validation-enabled.

Validation is specificed by including a `validate` section in your property definition, e.g. for `Age` in the example above. A validation section can contain at most one rule for each supported validation type, but usually will contain just one rule altogether.

Validation Type	Arguments	Description	Example
`int`	[min, max]	The property value must be an integer. If min is set, the property must be equal to or greater than the min. If max is set, the property must be less than or equal to the max. If min or max is set to `undef`, then no limiting is enforced	between 0 and 50, incl: 'int' => [0, 50] non-negative integer: 'int' => [0, undef]
`int-or-empty`	[min, max]	Available in NMIS 8.6.3G and newer. Like `int` except that an empty input is also accepted; this type is useful for properties that have global defaults.	
`float`	[min, max, above, below]	The property value must be a floating point number min and max work the same as for `int`. above sets an **exclusive** lower limit: the property must be strictly greater than above. below sets an **exclusive** upper limit. Any criteria set to `undef` are skipped.	between 0 and 1, incl: 'float' => [0, 1, undef, undef] greater than 0.1: [undef, undef, 0.1, undef]

`float-or-empty`	[min, max, above, below]	Available in NMIS 8.6.3G and newer. Like `float` except that an empty input is also accepted; this type is useful for properties that have global defaults.	
`regex`	a Perl regular expression given in `qr/.../` format	The property value must be matched by the regular expression.	a blank string or a +HH:MM/-HH:MM timezone: 'regex' => qr/^([+-]?\d{1,2}(:\d{1,2})?)?$/
`regex-or-empty`	a Perl regular expression given in `qr/.../` format	Available in NMIS 8.6.7G and newer. The property value must either be empty or match the given regular expression.	
`ip`	[acceptable IP versions]	The property must be a valid IP address. The rule argument sets which IP versions are acceptable; it can contain 4, 6 or both.	any IP address: 'ip' => [4, 6] an IP V4 address: 'ip' => [4]
`resolvable`	[acceptabe IP versions]	The property must be either a valid IP address, or it must be resolvable to a valid IP address (at the time of validation). Resolving is performed using the normal system mechanisms, ie. whatever combination of DNS and /etc/hosts is setup using `nsswitch`.	something with an IP V4 address: 'resolvable' => [4] something that resolves to an IP address: 'resolvable' => [4, 6]
`onefromlist`	[list of acceptable values] or `undef`	The property value must be one of the given explicit values, or one of the default display values if no values are given in this rule.	exactly one of the values that was presented visually using `value`: 'onefromlist' => undef one of these: 'onefromlist' => ['yes', 'no', 'maybe']
`multifromlist`	[list of acceptable values] or `undef`	Like `onefromlist`, but accepts multiple values from the accepted list. No values whatsoever does satisfy this validation rule.	zero or more of the presented values: 'multifromlist' => undef zero or more of these values: 'multifromlist' => [1, 2, 3, 4, 'OMGitsfulllofstars']

Management of Tables

Adding a New Table to NMIS

The following steps are required to add a new table:

1. Create a table configuration
2. Add the table to Tables.nmis
3. Create permissions in Access.nmis
4. Link to any other data

Create a Table Configuration

Create a file in /usr/local/nmis8/conf/, in our case /usr/local/nmis8/conf/Table-SampleTable.nmis, and add the appropriate configuration.

Add the table to Tables.nmis

Using the GUI or from the Unix prompt add the new table to Tables.nmis.

To add using GUI, access the menu item "System -> System Configuration -> Tables", a dialog will appear, and click on "add" next "Action >" in the top right of the widget.

Enter the properties for the table, the "Table Name" must match the name in the "Table Configuration", in our case SampleTable, the "Display Name" is what you want it to appear in the menu, and Description helps you remember what the table is for.

Refresh the NMIS Dashboard and your new table will exist in the menu but you will not be able to access it yet because there are no permissions defined for the table.

Create permissions in Access.nmis

You need to tell NMIS what Access permissions to add this with, we have created a script to add the tables with the default permissions which is as described in the table below, the command to run to add the permissions is.

```
/usr/local/nmis8/admin/add_table_auth.pl SampleTable
```

You should get some output like this:

```
Checking NMIS Authorisation for SampleTable
INFO: Authorisation NOT defined for SampleTable RW Access, ADDING IT NOW
INFO: Authorisation NOT defined for SampleTable View Access, ADDING IT NOW
```

The script can be run multiple times, it will not add the table twice.

The following table is the default permissions your table will be added with, if you want to change them, you can do that through the Access menu item at "System -> System Configuration -> Access Policy".

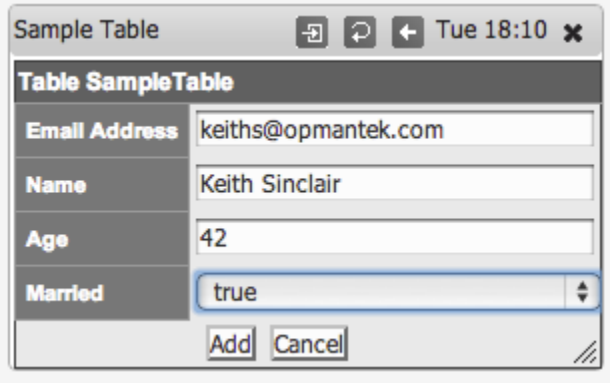
Level	Privilege	View	Read/Write
level0	administrator	Yes	Yes
level1	manager	Yes	Yes
level2	engineer	Yes	Yes
level3	operator	Yes	No
level4	guest	No	No
level5	anonymous	No	No
level6	security	No	No

* This step is intentionally done using the Unix shell, as we want to ensure that people adding privileges are truly NMIS admins and not someone sneaking up and using a browser window.

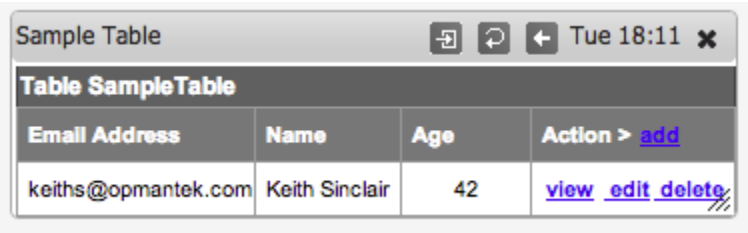
View the Table and Add Something

If you haven't already, refresh the NMIS Dashboard and access the new table through the menu, in this example "System -> System Configuration -> Sample Table". It will likely have an error message like "Error on loading table SampleTable" this is because there was not data.

So lets add some data, and the file will be created for us automatically.



Click on Add to save it and view the Table.



Linking Data Between Tables

Creating new tables isn't that thrilling but if we could start linking data between them, e.g. a select (drop down) in the Nodes table could contain information from a new custom table, then we would have a much more useful system for adding properties. Custom tables allow us to do this, as an example lets add a look up (displayed as a drop down) to our SampleTable called "Business Service".

To add a "Business Service" to our Sample table we will need to edit the Table Configuration and add some additional code to use the NMIS API (for looking up the values for "Business Service".

```
use NMIS;
use Auth;
my $C = loadConfTable();
# variables used for the security mods
my $AU = Auth->new(conf => $C); # Auth::new will reap init values from NMIS::config
# Calling program needs to do auth, then set the ENVIRONMENT before this is called.
$AU->SetUser($ENV{'NMIS_USER'});

%hash = (
  SampleTable => [
    { Email => { header => 'Email Address', display => 'key,header,text', value => [""] }},
    { Name => { header => 'Name', display => 'header,text', value => [""] }},
    { Age => { header => 'Age', display => 'header,text', value => [""] }},
    { Married => { header => 'Married', display => 'popup',value => ["true", "false"] }},
    { businessService => { header => 'Business Service', display => 'header,pop', value => [ sort keys %
{loadGenericTable('BusinessServices')} ] }},
  ]
);
```

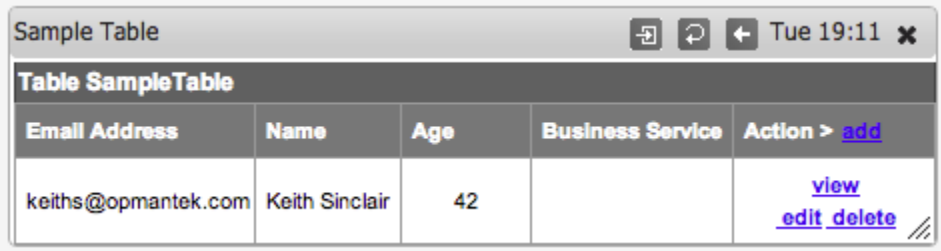
These lines setup the NMIS API

```
use NMIS;
use Auth;
my $C = loadConfTable();
# variables used for the security mods
my $AU = Auth->new(conf => $C); # Auth::new will reap init values from NMIS::config
# Calling program needs to do auth, then set the ENVIRONMENT before this is called.
$AU->SetUser($ENV{'NMIS_USER'});
```

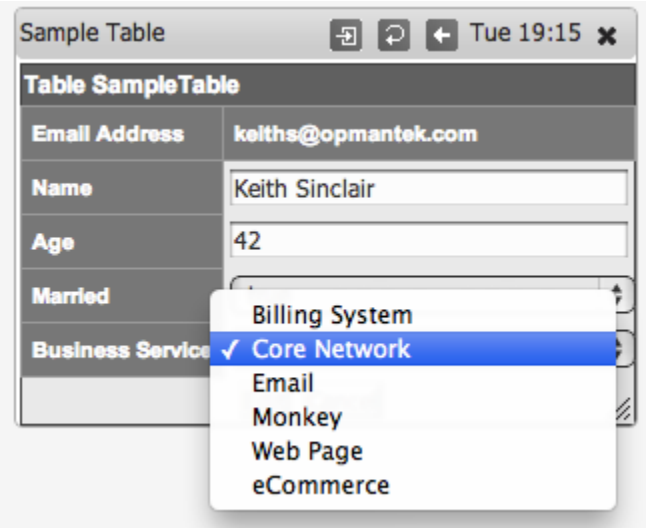
Then this line added to the %hash section gives up the lookup value. loadGenericTable('TableName') is what grabs the values to be displayed in the drop down for us..

```
 { businessService => { header => 'Business Service',display => 'header,pop',value => [ sort keys %
{loadGenericTable('BusinessServices')} ] }},
```

Refresh our widget and you will see the new empty value.



Edit that record and you can see a select box made up of the linked data.



Yes it really is that easy.

Troubleshooting

If you want to troubleshoot what is happening with a table when it is not working, you can look in the nmis log file, which is /usr/local/nmis8/logs/nmis.log, for example, refresh the table and then using the log tool select "NMIS_Log", or from unix "tail -50 /usr/local/nmis8/logs/nmis.log".

You might see an error like below, this is a Perl error and can be read the same way as reading a Perl compile or runtime error. Everything after the
 is the error. The information before the
 includes the callstack.

```
16-Mar-2013 13:48:26,tables.pl::loadCfgTable#140NMIS::loadGenericTable#293NMIS::loadFileOrDBTable#288func::
loadTable#869func::readFiletoHash#976<br>ERROR convert /usr/local/nmis8/conf/Table-Nodes.nmis to hash table,
Global symbol "$example" requires explicit package name at (eval 46) line 34, <$handle> line 70.
```

In this example 'Global symbol "$example" requires explicit package name at (eval 46) line 34' is saying there is a compile time error at line 46 of the evaluated code, in this case the variable $example has not been declared inline with Perl "strict".

The callstack looks like this:

```
NMIS::loadGenericTable (line 293)
  called:  NMIS::loadFileOrDBTable (line 288)
  called:  func::loadTable (line 869)
  called:  func::readFiletoHash (line 976)
```

Feedback

We would love you get your feedback, please let us know if you had any problems or would like more information at contact@opmantek.com