Node Administration Tools

NMIS provides a number of different methods for managing your nodes, both GUI-based and commandline-oriented. This document describes the commandline tools present in versions 8.5.4 and newer.

Import nodes to NMIS9 from NMIS8

To import nodes from NMIS8 to NMIS9 copy /path/to/nmis8/conf/Nodes.nmis to /tmp/ on NMIS9 installation then:

```
/usr/local/nmis9/admin/node_admin.pl act=import_bulk nodes="/tmp/Nodes.nmis"
```

Caveats: you can not import a node to a server if there is already a node existing with that name, you should rename the node before importing.

Bulk Import and Export

For importing lots of nodes in one go from a CSV file, NMIS has been providing `admin/import_nodes.pl` for a long time. There is also a sibling `admin/export_nodes.pl` tool, and both are documented on the Bulk Import page. The main benefit of these tools is utmost simplicity, but at the cost of some flexibility: `import_nodes` does not make all common node properties controllable or accessible.

Node administration with `node_admin.pl`

Version 8.5.4G brings in a new, more fine-grained and capable tool: admin/node_admin.pl. It's scriptable and pipelineable, and can perform all node-related operations: creation, updating, renaming, exporting and deletion of nodes.

Basic Operation

Run the tool with no options or -? or -h and it'll display a simple help page:

```
./admin/node_admin.pl -h
Usage: node_admin.pl act=[action to take] [extras...]
        node_admin.pl act=list
        node_admin.pl act={create|export|update|delete} node=nodeX
        node_admin.pl act=mktemplate [placeholder=1/0]
        node_admin.pl act=rename old=nodeX new=nodeY
mktemplate: prints blank template for node creation,
 optionally with __REPLACE_XX__ placeholder
create: requires file=NewNodeDef.json
export: exports to file=someFile.json (or STDOUT if no file given)
update: updates existing node from file=someFile.json (or STDIN)
delete: only deletes if confirm=yes (in uppercase) is given
extras: deletedata=<true,false> which makes delete also
delete all RRD files for the node. default is false.
extras: conf=<configname> to use different configuration
extras: debug={1..9,verbose} sets debugging verbosity
extras: info=1 sets general verbosity
```

Node listing and exporting

Given the argument `act=list`, `node_admin` will simply print a list of all known node names.

To see or save a node's information, run `admin/node_admin.pl act=export node=`*`somenodename`*, and it'll print the node's configuration in JSON format to your console. If you want to save that data in a file, either add the argument `file=`*`somefilename`*`.json` or redirect the output with `> somefile`. Here is an example of what to expect:

```
./admin/node_admin.pl act=export node=mytestbox
{
   "active" : "true",
   "businessService" : "my own test system",
   "collect" : "true",
   "community" : "verySecETr",
   "group" : "HQDev",
   "host" : "mytestbox.opmantek.com",
   "location" : "office",
   "model" : "automatic",
   "name" : "mytestbox",
   "netType" : "lan",
   "notes" : "there are no notes for this baby",
   "ping" : "true",
   "roleType" : "access",
   "version" : "snmpv2c"
}
```

Node Updating

Naturally `node_admin` does not just export node data but also consumes it for modifying a node in place and for creation of new nodes.

To change a node's configuration (**except** node renaming!), simply dump the node configuration with `act=export`, then edit the node with `act=update`. Both require that you give the node name in question, and both work either from files (with a `file=`*`somefile`*`.json` argument), or via STDOUT/STDIN /pipeline.

For example, this pipelined invocation would change the node from the example above to a less misspelled community:

```
./admin/node_admin.pl act=export node=mytestbox | sed -e 's/verySecETr/veryVerySecret/' | ./admin/node_admin.pl
act=update node=mytestbox
```

You can also use `file=-` to indicate that STDOUT should be used for export or STDIN be used for update/creation. The `act=update` operation doesn't create new nodes, and it replaces the whole set of node configuration settings with your new configuration input.

Creation of Nodes

The creation of nodes requires you to use a template (shown below) instead of using a command line argument; this is because NMIS requires numerous node properties to be set up correctly making it easy to miss some when operating via command line arguments. Node creation is triggered by the argument `act=create`, which behaves mostly like `act=update`, except that it doesn't touch existing nodes. To help you with starting a node configuration document from scratch (or in a scripted fashion), there is another command, `act=mktemplate`, which prints a blank but documented template which you can save and fill in. If you add `placeholder=1` to the command line, then `node_admin` fills the template with easily matchable replacement placeholders, like so:

```
./admin/node_admin act=mktemplate placeholder=1
// ... some comments
{
   "active" : "__REPLACE_ACTIVE__",
   "collect" : "__REPLACE_COLLECT__",
```

This makes it very easy to fill in the template with a script or some other external tool.

To create a node using this format start with:

1. Run the command: ./node_admin.pl act=mktemplate placeholder=1
   The results of this command are shown below:

```
{
   "active" : "__REPLACE_ACTIVE__",
   "collect" : "__REPLACE_COLLECT__",
   "community" : "__REPLACE_COMMUNITY__",
   "group" : "__REPLACE_GROUP__",
   "host" : "__REPLACE_HOST__",
   "location" : "__REPLACE_LOCATION__",
   "model" : "__REPLACE_MODEL__",
   "name" : "__REPLACE_NAME__",
   "netType" : "__REPLACE_NETTYPE__",
   "notes" : "__REPLACE_NOTES__",
   "ping" : "__REPLACE_PING__",
   "roleType" : "__REPLACE_ROLETYPE__",
   "version" : "__REPLACE_VERSION__"
}
```

2. Edit the information inside the template (i.e. change "__REPLACE_ACTIVE__" to "true") to correspond with the node you want to create then save it as a .json file.
3. Once the .json file is created and edited to suite then you run this command to create the new node: ./node_admin.pl act=create node=*newnode* file=*newnode.json* (replace "*newnode*" with a node name of your choice)
4. If the node was created successfully you should see a confirmation message saying, "Succesfully created node *newnode*.". To ensure the node was added you can go to the NMIS GUI and view it there as well.

Node Renaming

To rename nodes you should use `act=rename` which requires both old and new node names with arguments `old` and `new`, respectively. This operation first changes the node name (which is the primary name the node is known to and displayed by NMIS, and which is **NOT** necessarily the hostname or ip address of the node), and then adjusts all files related to the node in question:

- all RRD database files are renamed,
- and the node-related state files in `/usr/local/nmis8/var` are also renamed.

Deleting Nodes

To remove a node (but not its historic data) simply run `node_admin` with the argument `act=delete node=`*`ripnode`*, plus the option `confirm=YES` (must be uppercase) to make node_admin actually perform the deletion.

This removes only the node configuration record but not RRD database files or state files in `var.` To delete these as well, you can add the option `deletedata=1` to the command, and all data related to this node will be removed permanently.

```
./node_admin.pl act=delete node=ripnode confirm=YES deletedata=1
Successfully deleted ripnode
```

Node Properties

NMIS uses a subset of the node properties of the commercial Opmantek tools. `node_admin.pl act=mktemplate` includes a very brief listing of the most essential ones from NMIS' perspective, and the Common Node Properties wiki page describes most of the important ones in greater detail.

NMIS9 Enhancements

The node admin tool in NMIS9 brings some enhancements.

Backup, Migrate or just play with a Node

Simple node export and import are described above, however, with NMIS9 you can backup a node and perform node migrations using the node_admin.pl tool, the functions to do this are dump and restore.

Dump (or Backup) a Node

Using the node_admin.pl tool you can dump a node including all database records and RRD files into a ZIP file.

```
/usr/local/nmis9/admin/node_admin.pl node=NODENAME act=dump everything=1 file=/tmp/NODENAME-dump.zip
```

This file would represent a backup of that node at this time. The file can then be used on another server to restore or could be used to restore the node on the same server.

Restoring a Node

To restore a node to the same poller you would not need to localise_ids option, if you wanted to copy/migrate the node to another server you would need to localise the ids so that the poller thinks it the node belongs to it.

```
/usr/local/nmis9/admin/node_admin.pl act=restore file=NODENAME-dump.zip localise_ids=1
```

Caveats: you can not restore a node to a server if there is already a node existing with that name, you should rename the node before dumping. This would include if the server was acting as a master and receiving the node from a remote poller.

Copy a Node

To make a duplicate node and start polling it, export the node, edit the json and import it, e.g.

```
/usr/local/nmis9/admin/node_admin.pl act=export node=NODENAME file=NODENAME.json
```

Edit NODENAME.json, change the display_name and name in the file and then import it

```
/usr/local/nmis9/admin/node_admin.pl act=import file=NODENAME.json
```

The node will be created with the name used in the name field.

NMIS9 Basic Operation

Run the tool with no options or -? or -h and it'll display a simple help page:

```
Usage: node_admin.pl act=[action to take] [extras...]

        node_admin.pl act={list|list_uuid} [node=X] [group=Y]
        node_admin.pl act=show node=nodeX
        node_admin.pl act={create|update} file=someFile.json
        node_admin.pl act=export [format=nodes] [file=path] {node=nodeX|group=groupY} [keep_ids=0/1]
        node_admin.pl act=import file=somefile.json
        node_admin.pl act=import_bulk {nodes=filepath|nodeconf=dirpath}
        node_admin.pl act=delete {node=nodeX|group=groupY}
        node_admin.pl act=dump {node=nodeX|uuid=uuidY} file=path [everything=0/1]
        node_admin.pl act=restore file=path [localise_ids=0/1]

        node_admin.pl act=set node=nodeX entry.X=Y...
        node_admin.pl act=mktemplate [placeholder=1/0]
        node_admin.pl act=rename old=nodeX new=nodeY [entry.A=B...]

mktemplate: prints blank template for node creation,
 optionally with __REPLACE_XX__ placeholder

create: requires file=NewNodeDef.json
update: updates existing node from file=someFile.json

export: exports to file=someFile (or STDOUT if no file given),
 either json or as Nodes.nmis if format=nodes is given
 uuid and cluster_id are NOT exported unless keep_ids is 1.

delete: only deletes if confirm=yes (in uppercase) is given,
 if deletedata=true (default) then RRD files for a node are
 also deleted.

show: prints a node's properties in the same format as set
 with option quoted=true, show adds double-quotes where needed
set: adjust one or more node properties

restore: restores a previously dumped node's data. if
 localise_ids=true (default: false), then the cluster id is rewritten
 to match the local nmis installation.

extras: debug={1..9,verbose} sets debugging verbosity
extras: info=1 sets general verbosity
```