MediaMatrix®

A Division of Peavey Electronics Corporation

NWare

External control user guide

Version 2.0.0.0

July 7, 2017

Scope

This guide describes how to use different protocols, such as PASHA and SNMP, to remotely control and monitor devices in an NWare project.

Contents

Chapter 1
Introduction

In This Chapter

Remote Control

Remote control is a core feature of MediaMatrix hardware products. NWare, for example, connects to different nodes, including NioNodes and nControl nodes, as remote control software.

There are two categories of remote control:

Native	Used to set up and control nodes directly using software designed for the purpose, i.e. NWare.
External	Comprises protocols, e.g. RATC, that you can use to control MediaMatrix devices from other control systems.

Native Control

NWare : Design

NWare : Design, or simply *NWare*, is the software application that you use to design projects for MediaMatrix devices. You can use it to remotely control devices and manage a number of tasks, including logging, device status monitoring, project deployment and scripting.

For more information, refer to the *NWare User Guide*.

NWare : Kiosk

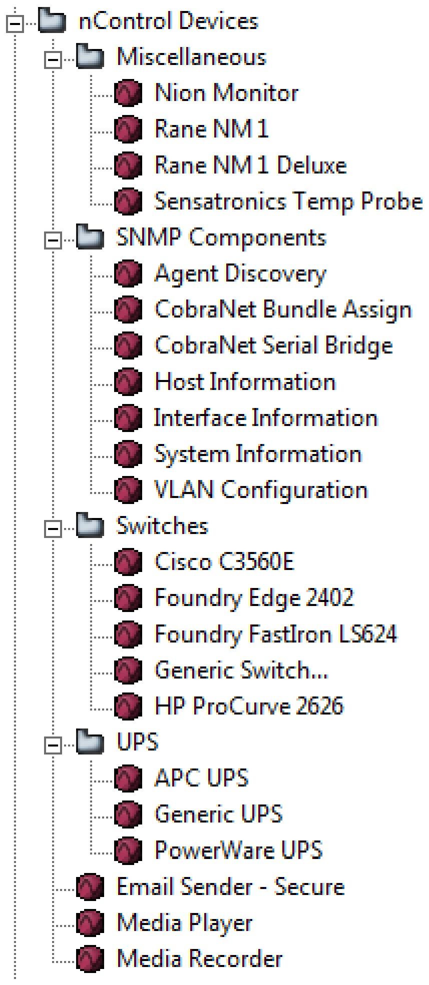
NWare:Kiosk or just *Kiosk*, is a control only application, which is launched by running the NWare application with the *personality* command line argument. Kiosk allows the user access to a design created in NWare, but not to change its configuration.

For more information, see *Running NWare in Kiosk mode* in the *NWare User Guide*.

nControl / nTouch 180 devices and SNMP

NWare includes devices specifically for use with nControl and nTouch 180 nodes. Communication with the majority of these devices takes place using SNMP. Since NioNodes do not have the ability to read and write SNMP values, these devices are specifically for nControl and nTouch 180 projects.

Tip: The NWare tree lists all the devices you can control using SNMP. If you cannot find a particular device, we may be able to provide you with a custom built device, as long as the uncompiled MIB file can be provided. The device can then be made available to other users in a future release of NWare.

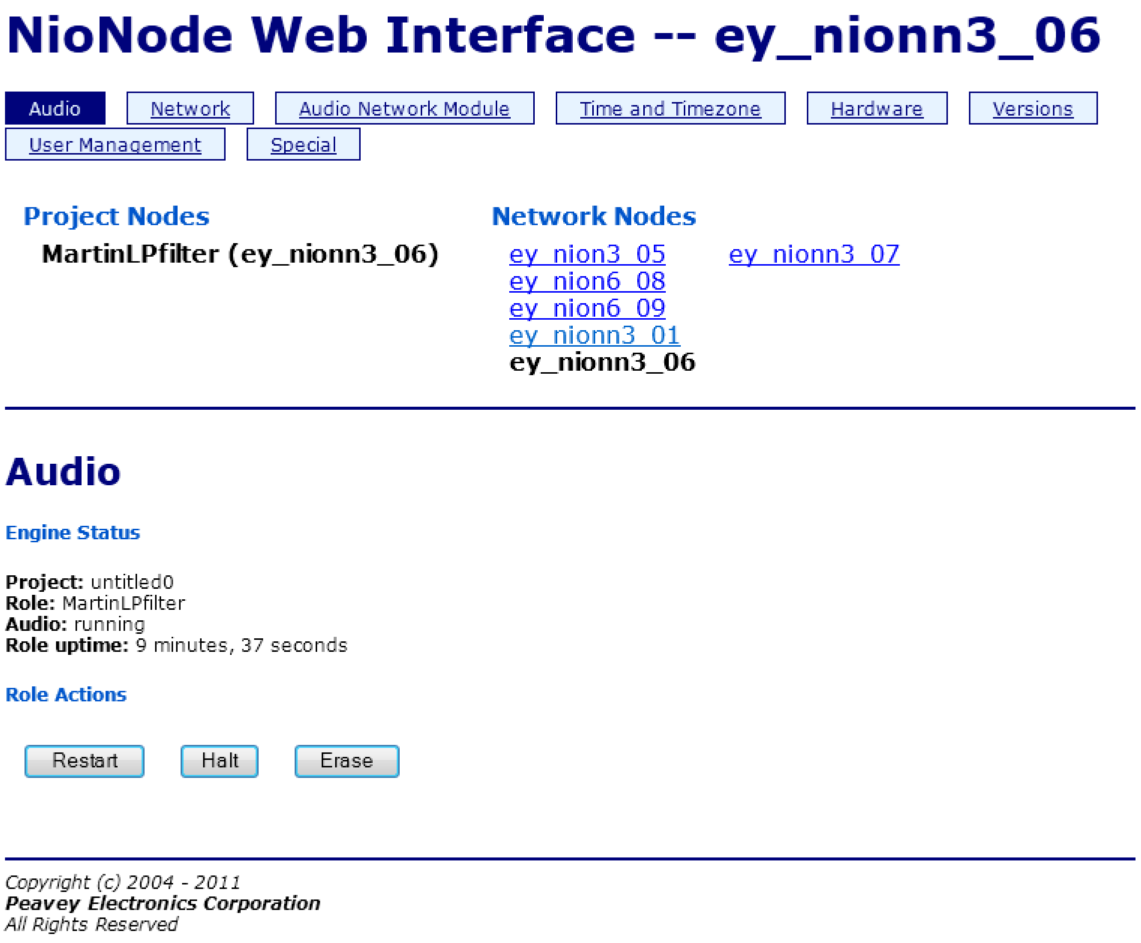


- nControl Devices
  - Miscellaneous
    - Nion Monitor
    - Rane NM 1
    - Rane NM 1 Deluxe
    - Sensatronics Temp Probe
  - SNMP Components
    - Agent Discovery
    - CobraNet Bundle Assign
    - CobraNet Serial Bridge
    - Host Information
    - Interface Information
    - System Information
    - VLAN Configuration
  - Switches
    - Cisco C3560E
    - Foundry Edge 2402
    - Foundry FastIron LS624
    - Generic Switch...
    - HP ProCurve 2626
  - UPS
    - APC UPS
    - Generic UPS
    - PowerWare UPS
  - Email Sender - Secure
  - Media Player
  - Media Recorder

NioNode web interface

Each NION has a built-in web server, which allows you to check the status, manage user accounts, update time and time zone settings, etc.

Note: We recommend that you restrict access to some features, including the **Audio** and **Network** pages, to prevent unauthorized users from changing settings. You can do this by changing the *defaultuser* account permissions from the **User Management** page of the web interface.



NioNode Web Interface -- ey_nionn3_06

Project Nodes

MartinLPfilter (ey_nionn3_06)

Network Nodes

ey_nion3_05
ey_nion6_08
ey_nion6_09
ey_nionn3_01
ey_nionn3_06
ey_nionn3_07

Audio

Engine Status

Project: untitled0
Role: MartinLPfilter
Audio: running
Role uptime: 9 minutes, 37 seconds

Role Actions

Copyright (c) 2004 - 2011
Peavey Electronics Corporation
All Rights Reserved

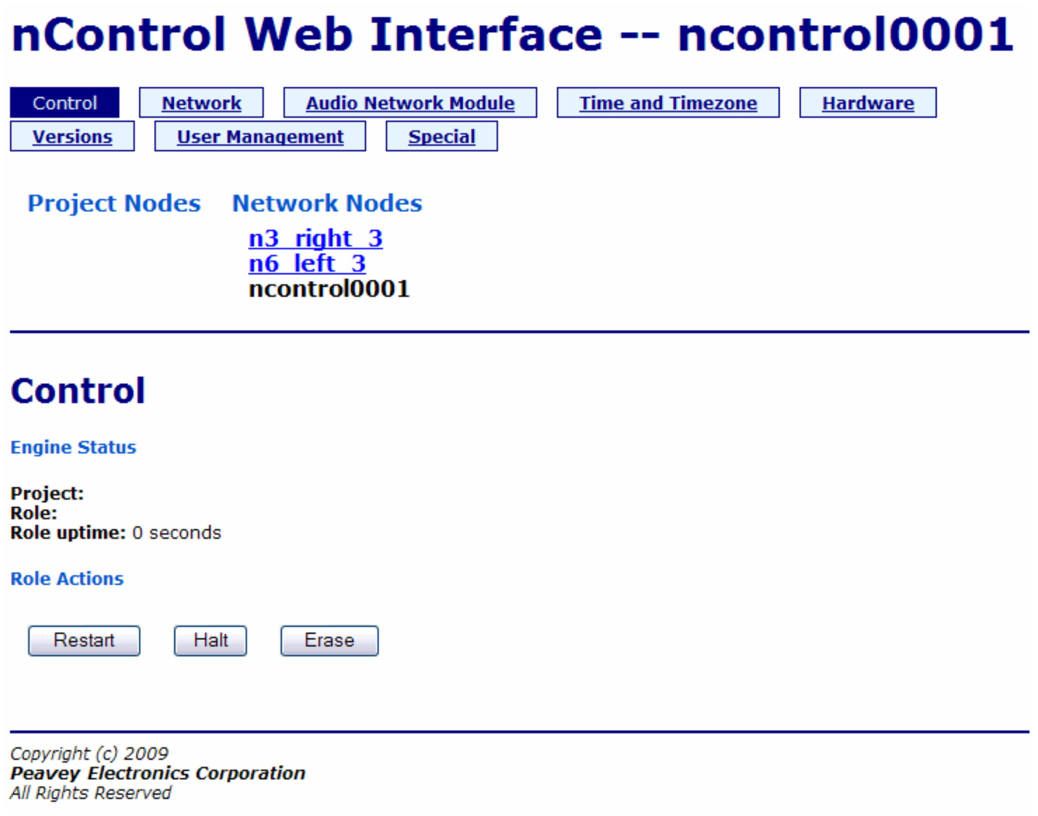
Tip: By clicking **Special**, and then **Advanced** from the screen shown above, you can view the screen that is currently shown on the NION front panel. The image can then be copied and pasted into your documentation and emails.

nControl web interface

Each nControl and nTouch 180 has a built-in web server, which allows you to check the status, manage user accounts, update time and time zone settings, etc.

Note: We recommend that you restrict access to some features, including the **Network** page, to prevent unauthorized users from changing settings. You can do this by changing the *defaultuser* account permissions from the **User Management** page of the web interface.

Here is an example screen from an nControl.



nControl Web Interface -- ncontrol0001

Project Nodes   Network Nodes
n3_right_3
n6_left_3
ncontrol0001

Control

Engine Status

Project:
Role:
Role uptime: 0 seconds

Role Actions

Copyright (c) 2009
Peavey Electronics Corporation
All Rights Reserved

External Control

RATC1 and RATC2

RATC is a command-line based protocol that allows a remote client program to set and get the control values in an NWare project. The remote client communicates with the project via a TCP network connection to any one of the NioNodes or nControl nodes in the project.

RATC1 is the first generation protocol that was used in Classic frame-based MediaMatrix systems. RATC1 for NioNodes, nTouch 180 nodes and nControl nodes is equivalent to what was called RATC in Classic MediaMatrix.

RATC2 is the improved version of RATC that was introduced with the NION platform, but is also supported by nControl and nTouch 180. Very similar to RATC1, RATC2 introduces shortened commands, and several extra functions.

PASHA

PASHA, the MediaMatrix Serial Handling Adapter, is a remote control protocol that provides external serial control and read-back of any of the controls appearing in an NWare project.

PASHA is a lower level and lower performance control than RATC1 or RATC2. It is primarily intended to be driven by programmable serial control devices.

Using multiple connector ports on the same NioNode

If you want to control devices via more than one serial port on the same NION, check the table below to see which combinations of protocols are valid.

	RS-422/485 serial port			
RS-232 serial port	**PASHA/PageMatrix**	**PASHA X/Control**	**RATC1**	**RATC2**
PASHA/PageMatrix	Yes	Yes	No	No
PASHA X/Control	Yes	Yes	No	No
RATC1	No	No	Yes	No
RATC2	No	No	No	Yes

Tip: You can use RATC1 or RATC2 on the Ethernet port and PASHA on one of the serial ports at the same time. Furthermore, this does not limit the character length of RATC commands.

Controlling NWare controls

In order to work with NWare controls using an external protocol like RATC, each control must be assigned a *control alias*. Once an alias has been assigned, you can write values to the control and read values from the control via a TCP/IP or serial connection, depending on the protocol you are using.

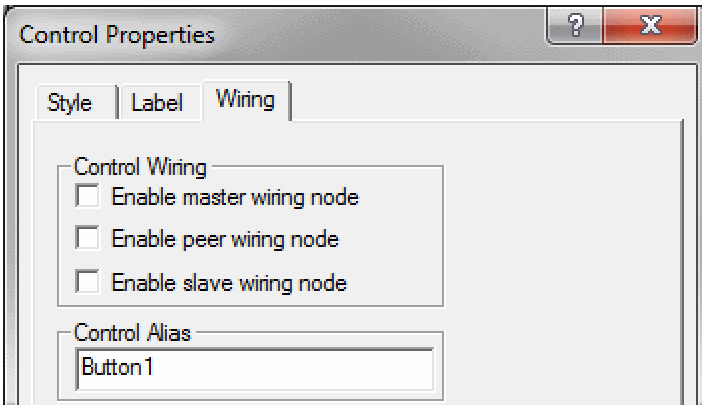
Note: When you specify an alias, NWare will check to see if that alias is already in use by another control. If it is, a warning will be displayed. If you click *Yes*, the control alias will be moved over from the old control to the new control. If you click *No*, the alias will not be moved. If you want to control multiple controls using the same alias, assign the alias to one of the controls and then use control wiring to connect them together in the project.

The external control system connects to the node hosting the project in order to access its controls. The controls are available via all of the nodes used in the project, so you can connect to any node in order to access a control. For redundancy support, however, we recommended that, rather than using a single node to access all the controls, you arrange for your system to failover to an alternative node should the original become unavailable.

Specifying a control alias on control properties

1. Right-click the control, and then click **Control Properties**.
2. Click the **Wiring** tab.
3. In the **Control Alias** box, type the alias that will be used to identify the control.

   In the example below, a control alias called *Button1* has been assigned to a control.



Button1

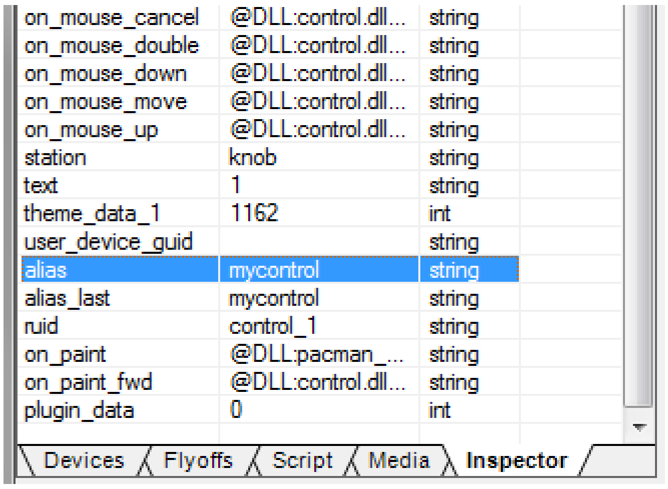
4. Click **OK**.

   NWare will check to see if that alias is already in use by another control. If it is, a warning will be displayed. If you click *Yes*, the control alias will be moved over from the old control to the new control. If you click *No*, the alias will not be moved.

Specifying a control alias using Inspector

1. If the Inspector tab is not shown in the NWare window, switch on Inspector.
2. Click the control you want to assign an alias.
3. Click the **Inspector** tab.
4. Scroll down to the bottom of the properties list.
5. Double-click in the box next to **alias**.

6. Type an alias name for the control and press Enter.



on_mouse_cancel	@DLL:control.dll...	string
on_mouse_double	@DLL:control.dll...	string
on_mouse_down	@DLL:control.dll...	string
on_mouse_move	@DLL:control.dll...	string
on_mouse_up	@DLL:control.dll...	string
station	knob	string
text	1	string
theme_data_1	1162	int
user_device_guid		string
alias	mycontrol	string
alias_last	mycontrol	string
ruid	control_1	string
on_paint	@DLL:pacman_...	string
on_paint_fwd	@DLL:control.dll...	string
plugin_data	0	int

Viewing NWare control aliases

Using RATC

You can list out all the control aliases you have assigned to controls in your NWare project. This feature is especially useful when you have a large number of controls and you want to see which ones have been assigned aliases.

If you are already using RATC, you can telnet to the NioNode or nControl node and issue the command. If you are not using RATC, you will need to enable it temporarily.

⏩ *To enable RATC*

1. Open NWare.
2. Right-click the NioNode or nControl node on the page and then click **Device Properties**.
3. In the **Network Control Protocol** list, click **RATC2**.
4. On the toolbar, click the **Deploy** button .

Tip: If the project is very large and contains many NioNodes, you can deploy to a single NioNode and emulate the others.

⏩ *To list NWare control aliases*

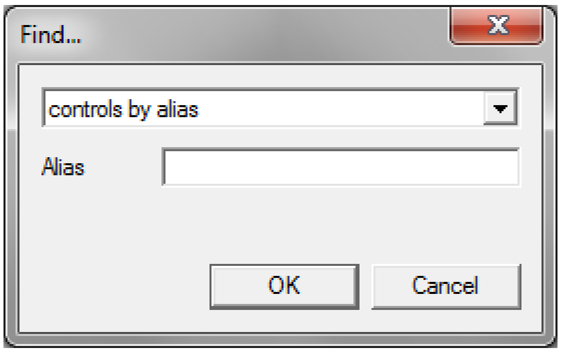
1. Open a telnet program and connect to the NioNode or nControl node using its IP address.
   If you are using Windows XP, you can open a Command Prompt window and type `telnet <IP address> 1632` to connect to the node.
2. Press Enter.
3. Type `controlList`.
   A list of all control aliases in the project will be displayed.

Tip: If the error message `notLoggedIn` is displayed, open the user properties for *defaultuser* in NWare. In the **Network Control Access** list, click **Allowed**. Redeploy the project and then refer to step 1 in this procedure.

Finding a control in a project using its alias

1. Press CTRL+F.

   The **Find** dialog box is displayed.



2. In the list, click **controls by alias**.
3. In the **Alias** box, type the name of the alias to search for.
4. Click **OK**.

   The search results will be displayed on the **Find Results** tab at the bottom of the NWare window.
5. Click a search result to jump to the device in the project.

Finding the control alias definitions inside the NWare project file

All the control aliases in a project are stored in a file called *control_alias.xml*. This file is kept inside the project *.npa* file, which is a zip file.

⏩ *To find the control alias definitions inside the NWare project file*

1. Open Windows Explorer, and then locate the NWare *.npa* project file.
2. Change the file extension to *.zip*.
3. Double-click the new *.zip* file to display its contents.
4. Locate the file *control_alias.xml* inside the zip file, and then open this file in an editor.

   Each of the control alias definitions looks like this:

   ```
   <alias alias="EQ1_BW" ruid="//devices/41/controls/bandwidth" />
   ```

   The alias name is *EQ1_BW* in this example. You can use the *NWare find function* (on page 13) to locate this control in the project using its alias.

   Some aliases are generated automatically, like this one for a Kiosk2Go control:

   ```
   <alias alias="X139753064" ruid="//devices/28/controls/message" />
   ```

Exporting control alias names into a CSV file

You can export a list of alias names that have been assigned manually to controls in a project. The list is created in CSV format.

Note: Names that are generated automatically by NWare are not included in the export.

⏩ *To export alias names into a CSV file*

1. On the **File** menu, click **Export Control Aliases**.
2. Choose a location and file name for the exported file.

3. Click **Save**.

Removing automatically assigned control aliases

In versions of NWare prior to 1.7.0b, all generic controls in a project were assigned alias names automatically when the project was deployed or emulated. This meant that when control aliases from a large project were listed out using a protocol like RATC, there could be hundreds of returned values.

You can use a feature in NWare to remove all the automatically assigned aliases (starting with X or ~) from the entire project. Only aliases you have assigned manually to controls will be retained.

Note: There is no way to retrieve these aliases once removed.

⏩ *To remove automatically assigned control aliases*

- On the **Tools** menu, click **Clean Project**.

Configuration and testing

TCP/IP and serial control services are configured in the properties of the NioNodes, nControl and nTouch 180 nodes in the NWare project.

You can test external control via TCP/IP and serial connections using the NWare Emulator.

Note: If you have enabled external control on more than one node, this may generate port conflict errors, as each emulated node shares the same ports on the NWare PC. As a workaround, while you are testing your project, change the TCP/IP or serial (COM) port numbers so that each node uses a different port number.

Chapter 2

RATC

In This Chapter

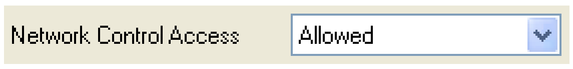
Introduction

RATC is compatible with both local area networks and the Internet. Any number of RATC clients may connect simultaneously to any MediaMatrix node in the project. The RATC1 or RATC2 service is configured and enabled through the node device properties in NWare.

Controls are made available to RATC1, RATC2 and/or PASHA using *control aliases*. PASHA only supports 3 character and 4 numeral aliases, but RATC1 and RATC2 support aliases of several dozen characters, including spaces.

Note: When you specify an alias, NWare will check to see if that alias is already in use by another control. If it is, a warning will be displayed. If you click *Yes*, the control alias will be moved over from the old control to the new control. If you click *No*, the alias will not be moved. If you want to control multiple controls using the same alias, assign the alias to one of the controls and then use control wiring to connect them together in the project.

You can enable RATC communications for a project on a per-user basis, via the user account properties. The user account properties can be displayed by clicking **User Accounts** on the **File** menu, selecting a user, then clicking **Edit**. The **Network Control Access** setting must be set to **Allowed** for RATC to be enabled.


Network Control Access Allowed

RATC versions

Classic MediaMatrix supported RATC v0.9 on TCP/IP connections. NION and nControl support the newer and more flexible protocol, RATC v2, which can run on either a TCP/IP connection or a serial connection.   We refer to the two versions of RATC as RATC1 and RATC2.

For backward compatibility, NION and nControl support RATC v1, which is compatible with MediaMatrix's RATC v0.9.   NION also features a version of PASHA that is compatible with the service of the same name on Classic MediaMatrix.

Note: PASHA is not currently available on nControl units.

It is recommended that RATC v2 over TCP/IP be used whenever possible.

One difference between NION and nControl external control and Classic MediaMatrix external control is that in NION and nControl systems, the external control services only run while the NWare project is running. This is a little different than Classic MediaMatrix, where the control services run whenever the MWare application is running, independent of whether a View file is compiled and running.

Using telnet with RATC

RATC1 can function as a telnet server. This means that any standard telnet client program may be used to manipulate and monitor control group values for MediaMatrix products. This is useful for verifying that the configuration will allow a successful RATC connection to the MediaMatrix device.

To connect a telnet terminal to RATC1, specify to the telnet client program the MediaMatrix device name or IP address and the RATC port number (usually 1632).

RATC does not echo data input from the client, so we advise that local echo is enabled on the telnet client program, so that you can see what you are typing while you type it.

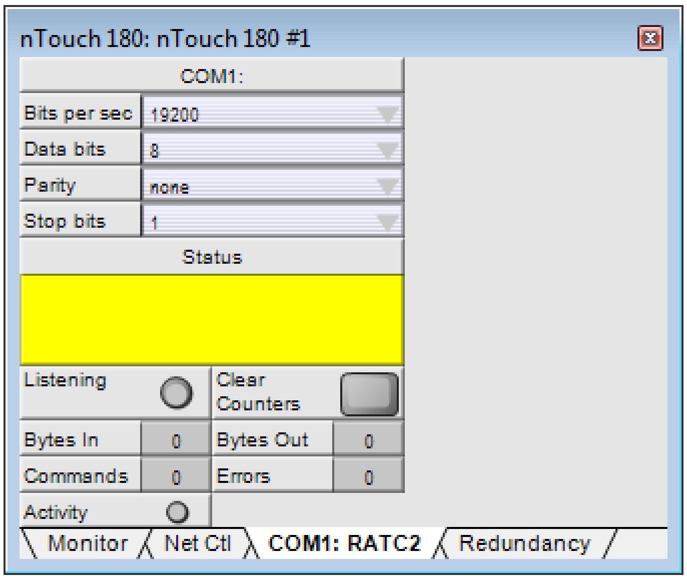
Notes:

- Although you can use a telnet style application to control MediaMatrix products with RATC, we recommend using custom software for real world applications. Telnet is handy for testing your RATC connection.
- The BEL character that precedes each error response from RATC may cause a bell sound on your telnet client computer.
- RATC supports only the minimum telnet protocol, and thus will refuse all telnet option requests from a telnet client program.

Allowing RATC to be used on a MediaMatrix node

1. Right-click the NioNode, nControl node or nTouch 180 node, and then click **Device Properties**.
2. If you are using a serial connection, click **RATC1** or **RATC2** in the **RS-232 Protocol (COM port)** list.

   If you are using a TCP/IP connection, click **RATC1** or **RATC2** in the **Network Control Protocol** list.
3. Deploy the project, and then specify settings on the **Serial RATC** or **Net Ctl** tab to match your control system. For example:



nTouch 180: nTouch 180 #1

COM1:

Bits per sec	19200
Data bits	8
Parity	none
Stop bits	1

Status

Listening		Clear Counters	
Bytes In	0	Bytes Out	0
Commands	0	Errors	0
Activity			

Commands and Responses

Each RATC1 command is an ASCII text string terminated with an ASCII CR. Each command results in a response from RATC1. Except during login, RATC1 will send no unsolicited data to a client. Each response from RATC1 is an ASCII text string terminated with both an ASCII CR and NL character. Each response that indicates an error in the input command line is preceded by an ASCII BEL character.

RATC1 commands are not case-sensitive, but the password string is case-sensitive (as per the MediaMatrix security model). RATC1 commands are, however, presented here with mixed case to improve readability.

Control codes other than CR in the command line are ignored, except for the following two exceptions: ASCII BS (backspace) is supported to make using telnet more reasonable; and the telnet option control escape sequence protocol is automatically dealt with (all telnet option requests are refused by the RATC1 service).

Name	**ASCII Name**	**C++ Name**	**Decimal Value**	**Hexadecimal Value**
carriage return	CR	\r	13	D
newline	NL or LF	\n	10	A
alert	BEL	\a	7	7
backspace	BS	\b	8	8

RATC1 commands and responses are designed to be compatible with two use scenarios: computer control with a software application and human control with telnet. To aid in the text parsing required for computer control, the non-error responses all have a unique first character, and the error responses all have an ASCII BEL character followed by a unique character.

Change groups

If the client software will be required to monitor a large number of control alias values, the *Change Group* commands can be used to maximize efficiency. By issuing a single command, you can detect whether any individual controls in a group of controls have changed.

When a control alias is added to the change group, it is considered initially *changed*, and you can retrieve its value by issuing a *get* command.   You can define multiple change groups using RATC2.

At the initial connect, and after each subsequent deploy or emulate, there will be no change groups, because they are cleared between deployments. If there are no change groups assigned when a *Change Group Get* command is received, a default change group will be created (but containing no controls), so the response would show zero changes. It is not necessary for the client software to clear the change group before disconnecting – this is done automatically by the MediaMatrix node.

RATC1 commands

help Command

Usage

As an aid in the telnet operation of RATC1, the following command will result in a helpful display listing the RATC1 command set

```
help\r
```

Response

The help response starts with

```
{
```

and the final line of the response is:

```
}\r\n
```

as an aid in allowing a computer program to ignore the contents of the help response.

Note: Parsing of the help and list command responses by a software-based client is strongly discouraged since the formats are subject to change.

Possible Error Messages

```
\aOverflow\r\n
```

statusGet command

Usage

Gets the state and name of the project running on this NioNode project member.
The client issues:

```
statusGet\r
```

Response

RATC1 responds with:

```
statusIs running "Level 1 Ballrooms"\r\n
```

If no project is running, you will not be able to connect, so there is no alternative response here (no "stopped or not running" status).

Note: The file name is enclosed in quotes to support spaces in the file name. In future versions, we plan to add other states for the second part of the response for other situations.

Possible Error Messages

```
\aOverflow\r\n
```

controlGet command

Usage

The *controlGet* command is used to determine the value of a Control Alias in a running project. For example:

```
controlGet "Main Gain"\r
```

Response

On success, RATC1 responds with:

```
valueIs "Main Gain" +0.00dB 70.0%\r\n
```

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\aOverflow\r\n
```

```
\aUnlistedGroup "AliasName"\r\n
```

In the context of NION and nControl, *UnlistedGroup* refers to the lack of a control with that control alias. In this instance, the term group is used for compatibility purposes with Classic MediaMatrix RATC.

controlSet command

Usage

The *controlSet* command is used to set the value of an aliased control in a deployed project. For example:

```
controlSet "Main Gain" -3.4\r
```

Response

On success, RATC1 responds with:

```
valueIs "Main Gain" -3.40dB 61.5%\r\n
```

The Control Alias name is enclosed in quotes to support names with embedded spaces. In fact, the quotes are not required for the command argument if the Control alias has no embedded spaces.

In the response, the first token after the alias name is the current value expressed in the units appropriate to that particular Control Alias. In the response, the second token after the Control Alias name is the current value of the alias expressed as a percentage of the maximum value. This can be thought of as a physical knob position - in this example, the 61.5% knob position corresponds to -3.4dB, and a 100% knob position will correspond to +12dB.

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\aOverflow\r\n
```

```
\aUnlistedGroup "GroupName"\r\n
```

In the context of NION and nControl, *UnlistedGroup* refers to the lack of a control with that control alias. In this instance, the term group is used for compatibility purposes with Classic MediaMatrix RATC.

controlList command

Usage

As an aid in the telnet operation of RATC1, the following command:

```
controlList\r
```

will result in a display listing the current set of control alias names.

Response

The list response starts with:

```
{
```

and the final line of the response is:

```
}\r\n
```

as an aid in allowing a computer program to ignore the contents of the list response.

Notes:

- There may be other matching brace characters within the list response.
- Parsing of the help and list command responses by a software-based client is strongly discouraged since the formats are subject to change.

Possible error messages

```
\aOverflow\r\n
```

changeGroupAddControl command

Usage

A Control Alias is added to the Change Group with the `changeGroupAddControl` command. For example:

```
changeGroupAddControl "Main Gain"\r
```

Response

On success, RATC1 responds with:

```
addedToChangeGroup "Main Gain"\r\n
```

Possible error messages

```
\aBadArgumentCount\r\n
```

```
\aOverflow\r\n
```

```
\aInvalidChangeGroup
```

```
\aUnlistedGroup "GroupName"\r\n
```

In the context of NION and nControl, 'UnlistedGroup' refers to the lack of a control with that control alias. In this instance, the term group is used for compatibility purposes with Classic MediaMatrix RATC.

changeGroupRemoveControl command

Usage

A Control Alias is removed from the Change Group with the *changeGroupRemoveControl* command. For example:

```
changeGroupRemoveControl "Main Gain"\r
```

Response

On success, RATC1 responds with:

```
removedFromChangeGroup "Main Gain"\r\n
```

The above response is issued even if the Control Alias named was not in the Change Group.

Possible error messages

```
\aBadArgumentCount\r\n
```

```
\aOverflow\r\n
```

```
\aInvalidChangeGroup
```

```
\aUnlistedGroup "GroupName"\r\n
```

In the context of NION and nControl, 'UnlistedGroup' refers to the lack of a control with that control alias. In this instance, the term group is used for compatibility purposes with Classic MediaMatrix RATC.

changeGroupClear command

Usage

The Change Group is cleared of all Control Aliases with the command:

```
changeGroupClear\r
```

Response

On success, RATC1 responds with:

```
clearedChangeGroup\r\n
```

Possible error messages

```
\aOverflow\r\n
```

changeGroupGet command

Usage

The Control Groups in the Change Group are polled for value changes with the command:

```
changeGroupGet\r
```

All values that have changed will be returned.

Response

On success, RATC1 responds with the number of values that have changed and a value for each change. For example:

```
numberOfChanges 2\r\n
```

```
valueIs "Main Gain" -15.2dB 46.1%\r\n
```

```
valueIs "Channel1Gain" -20.0dB 50.0%\r\n
```

In this example, there are two Control Aliases in the Change Group that have changed since the last query. If the command is issued again, with no changes to Control Aliases, or if there are no Control Aliases in the Change Group, the response will be:

```
numberOfChanges 0\r\n
```

Possible error messages

```
\aBadArgumentCount\r\n
\aOverflow\r\n
\aNotRunning\r\n
\aInvalidChangeGroup
```

RATC1 responses

```
statusIs
valueIs
addedToChangeGroup
removedFromChangeGroup
clearedChangeGroup
numberOfChanges
```

There are also various error responses.

In addition, the RATC1 login process uses a name prompt, a password prompt, a version statement and a welcome statement.

Chapter 3

RATC2

In This Chapter

Introduction

RATCv2 is telnet compatible, meaning that it is text-based and that it is possible to use a telnet client program to drive it.

If you want to send RATC2 commands over TCP, each command and its arguments must be followed by a CR and/or LF. However, if you are sending RATC2 commands over a serial connection, only an LF must be used.

Note: In contrast to RATC1 commands, RATC2 commands are case sensitive.

Each response is followed by CR/LF. The command is a token, the alias is a token, and the value is a token. The alias can have spaces, but if it does, it must be enclosed in double quotes so as to make it a single token.

Each command has an abbreviated shortcut formed from the first letter of each word within the command.

Command list

command	**short version**	**purpose**
help	h	display the list of commands
logIn name password	li	log in with username and password
statusGet	sg	report status
keepAlive seconds	ka	disconnect if no activity in n seconds
quietModeEnable	qme	suppress responses from non-query commands
quietModeDisable	qmd	do not suppress responses
controlGet control	cg	get a Control's value and position
controlSet control value	cs	set a Control's value
controlPositionSet control position	cps	set a Control's position (0.00-1.00)
changeGroupControlAdd [group] control	cgca	add a Control to a Change Group
changeGroupControlRemove [group] control	cgcr	remove a Control from a Change Group

command	short version	purpose
changeGroupGet [group]	cgg	get changed values from a Change Group
changeGroupClear [group]	cgc	clear a Change Group (of changed values)
changeGroupSchedule [group] seconds	cgs	schedule recurring Change Group gets **Note:** This command is only supported over TCP connections.

RATC2 responses

```
statusIs
valueIs
loggedIn
keepAlive
quietModeEnabled
quietModeDisabled
changeGroupControlAdded
changeGroupControlRemoved
changeGroupCleared
changeGroupChanges
changeGroupSchedule
```

RATC2 error responses

The complete list of error responses is as follows:

```
badCommand
badArgumentCount
overflow
unlistedControl
invalidChangeGroup
commandFailed
commandUnsupported
notLoggedIn
loginFailed
```

Note: The RATC2 login process does not display a username and password prompt. To login, use the *li* command.

Commands in detail

help Command

Command	help
Shortcut	h
Arguments	none
Availability	always
Purpose	displays a list of commands
Notes	The number of line/commands to expect is given in the first line
Response	a list of the available commands

Usage example

```
help\r
```

or

```
h\r
```

Response

```
RATCv2.0 Help 15
h    help : display this help list
li   logIn name password : log in with a password
sg   statusGet : report status
ka   keepAlive seconds : disconnect if no activity in n seconds
qme  quietModeEnable  : suppress responses from non-query commands
qmd  quietModeDisable : allow responses from all commands
cl   controlList : get the list of available Controls
cg   controlGet control : get a Controls value
cs   controlSet control value : set a Controls value
cps  controlPositionSet control value : set a Controls position
(0.00-1.00)
cgca changeGroupControlAdd [group] control : add a Control to a Change
Group
cgcr changeGroupControlRemove [group] control : remove a Control from
a Change Group
cgg  changeGroupGet [group] : get changed values from a Change Group
cgc  changeGroupClear [group] : clear a Change Group (of changed values)
cgs  changeGroupSchedule [group] seconds : schedule recurring Change
Group gets
```

Note: Parsing of the help and list command responses by a software-based client is strongly discouraged since the formats are subject to change.

Possible Error Messages

```
\aOverflow\r\n
```

login command

Command	logIn
Shortcut	li
Arguments	<username> <password> If password is blank, it can be omitted. If both username and password are blank, both can be omitted.
Availability	always
Purpose	security
Notes	you are not prompted to log on, but must instead explicitly issue the login command. A few commands are available prior to logging in: help, quietModeEnable, quietModeDisable. If any other command is attempted prior to logging in, the response will be notLoggedIn.
Response	loggedIn or loginFailed

Usage example

If the client's username is *maintenance* and the password is *youguessedit*, the client should type:

```
logIn maintenance youguessedit\r
```

or

```
li maintenance youguessedit\r
```

Response

```
loggedIn
```

Possible error messages

```
\aloginFailed\r\n
```

```
\aOverflow\r\n
```

statusGet command

Command	statusGet
Shortcut	sg
Arguments	none
Availability	always
Purpose	get the current state of the system
Response	Something like this: statusIs running "Your Project Here"

Usage Example

The client issues:

```
statusGet\r
```

or

```
sg\r
```

Response

```
statusIs running "Level 1 Ballrooms"\r\n
```

If no project is running, you will not be able to connect, so there is no alternative response here (no "stopped or not running" status).

Note: The file name is enclosed in quotes to support spaces in the file name. In future versions, we plan to add other states for the second part of the response for other situations.

Possible Error Messages

```
\aOverflow\r\n
```

keepAlive command

Command	keepAlive
Shortcut	ka
Arguments	seconds
Availability	when logged in
Purpose	Starts a watchdog timer, requiring the client to communicate within that period or be disconnected. It is recommended that keepAlive be used to ensure that TCP/IP client connections get closed in the face of network failures (or even just unplugging and plugging in network connections). If the network goes down and the client attempts to communicate, the client will get a timeout and disconnect, reconnecting once the network is restored. The server, however, would normally have no way of knowing that the client gave up on its first connection and reconnected, so server network stack resources consumed over time. keepAlive solves this problem.
Notes	You can disable the watchdog timer with an argument of zero. This command applies to TCP/IP only, not to the serial port.
Response	Something like this: keepAlive 10

Usage Example

The client issues:

```
keepAlive 120\r
```

or

```
ka 120\r
```

Response

```
keepAlive 120\r\n
```

Possible Error Messages

```
\aOverflow\r\n
\aBadArgumentCount\r\n
\anotLoggedIn\r\n
```

quietModeEnable command

Command	quietModeEnable
Shortcut	qme
Arguments	none
Availability	always
Purpose	Suppresses the server responses to some of the commands: logIn, controlSet, changeGroupControlAdd, changeGroupControlRemove, changeGroupClear, keepAlive, and quietModeEnable.
Notes	
Response	none!

Usage Example

The client issues:

```
quietModeEnable\r
```

or

```
qme\r
```

Response

RATC2 does not respond.

Possible Error Messages

```
\aOverflow\r\n
```

quietModeDisable command

Command	quietModeDisable
Shortcut	qmd
Arguments	none
Availability	always
Purpose	To leave quiet mode, restoring responses to all commands
Response	quietModeDisabled

Usage Example

The client issues:

```
quietModeDisable\r
```

or

```
qmd\r
```

Response

```
quietModeDisabled\r\n
```

Possible Error Messages

```
\aOverflow\r\n
```

controlGet command

Command	controlGet
Shortcut	cg
Arguments	control alias
Availability	when logged in
Purpose	To read the current value and position of a control
Notes	This command returns the control alias name, the string value of the control (such as "6.2dB"), and the positional value (0.000 through 1.000). When the control name has spaces, the argument should be enclosed in double-quotes, as in: controlGet "Master Gain"
Response	Something like this: valueIs "Master Gain" -90.0dB 0.100

Usage Example

The client issues:

```
controlGet "Main Gain"\r
```

or

```
cg "Main Gain"\r
```

The Control Alias name is enclosed in quotes to support names with embedded spaces. In fact, the quotes are not required for the command argument if the Control name has no embedded spaces.

Response

```
valueIs "Main Gain" 1.99dB 0.864\r\n
```

Note: The positional scale of 0.864 is on a 0.000 to 1.000 scale, but can be read as 86.4% of the control's range. The first value after the alias name is the exact reading of the control (known as the "string value") and will differ, depending on the control type, however, all responses use this 0.000 to 1.000 scale for the last element of the response.

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\anotLoggedIn\r\n
```

```
\aUnlistedControl "AliasName"\r\n
```

```
\aOverflow\r\n
```

controlSet command

Command	controlSet
Shortcut	cs
Arguments	control value
Availability	when logged in
Purpose	To set the string value of a control
Notes	Like controlGet, this command returns the Control name, the string value of the Control, and the positional value. When the Control name has spaces, the argument should be enclosed in double-quotes, as in: controlSet "Master Gain" -75
Response	Something like this: valueIs "Master Gain -75.0dB 0.250

Usage Example

The client issues:

```
controlSet "Main Gain" -3.4\r
```

or

```
cs "Main Gain" -3.4\r
```

to set the control value to -3.4.

The Control Alias name is enclosed in quotes because it contain spaces. (If the name does not contain spaces, the quotes are not required.)

You can adjust the value of the control relative to its current value using:

```
cs "Main Gain" ++3\r
```

or

```
cs "Main Gain" --3\r
```

These examples increase the value by 3dB and decrease it by the same amount respectively.

You can reset the value of a string control by setting its value to double quotes "". For example:

```
controlSet "strcontrol2" ""
```

Response

```
valueIs "Main Gain" -3.50dB 0.818\r\n
```

In the response, the first token after the alias name is the current value expressed in the units appropriate to that particular control. In the response, the second token after the Control Alias name is the current value of the alias expressed as a percentage of the maximum value expressed as a decimal value between 0 and 1. This can be thought of as a physical knob position - in this example, the 0.615 (61.5%) knob position corresponds to -3.4dB, and a 1.000 (100%) knob position will correspond to +12dB.

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\anotLoggedIn\r\n
```

```
\aUnlistedControl "AliasName"\r\n
```

```
\aOverflow\r\n
```

controlPositionSet command

Command	controlPositionSet
Shortcut	cps
Arguments	control position.
Availability	when logged in
Purpose	to set the positional value of a control, corresponding to a "slider position" between 0 and 1. For example, .535 is a position of 53.5%.
Notes	Like controlGet and controlSet, this command returns the control name, the string value and the positional value of the control. When the control alias has spaces, the argument should be enclosed in double-quotes, as in: controlSet "Master Gain" .25
Response	Something like this: valueIs "Master Gain" -75.0dB 0.250

Usage Example

The client issues::

```
controlPositionSet "Master Gain" .25\r
```

or

```
cps "Master Gain" .25\r
```

Response

```
valueIs "Master Gain" -75.0dB 0.250\r\n
```

Possible Error Messages

```
\aBadArgumentCount\r\n
\anotLoggedIn\r\n
\aOverflow\r\n
```

controlList Command

Command	controlList
Shortcut	cl
Arguments	none
Availability	when logged in
Purpose	to list controls that can be accessed
Response	presents a list of all control aliases in double-quotes with CR/LF after each

Usage Example

The client issues:

```
controlList\r
```

or

```
cl\r
```

which results in a response listing the current set of Control Alias names.

Response

If the aliases are: "control1 ... control5", then the response would be:

```
"control1"\r\n
"control2"\r\n
"control3"\r\n
"control4"\r\n
"control5"\r\n
```

Note: Parsing of the help and list command responses by a software-based client is strongly discouraged since the formats are subject to change.

Possible Error Messages

```
\anotLoggedIn\r\n
```

```
\aOverflow\r\n
```

changeGroupControlAdd command

Command	changeGroupControlAdd
Shortcut	cgca
Arguments	[group] control
Availability	when logged in
Purpose	To add a control to a Change Group. If the Change Group named does not yet exist, it is created.
Notes	The group name argument is optional. If not included, a Change Group named 'default Change Group' will be used.
Response	changeGroupControlAdded

Usage example

The client issues:

```
changeGroupControlAdd "Main Gain"\r
```

or

```
cgca "Main Gain"\r
```

Response

```
changeGroupControlAdded\r\n
```

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\anotLoggedIn\r\n
```

```
\aUnlistedControl "AliasName"\r\n
```

```
\aOverflow\r\n
```

changeGroupControlRemove Command

Command	changeGroupControlRemove
Shortcut	cgcr
Arguments	control [group].
Availability	when logged in
Purpose	to remove a control from a Change Group.
Notes	The group name argument is optional. If not included, a Change Group named 'default Change Group' will be used. If the Change Group named does not exist, the response is something like invalidChangeGroup "yourBogusGroupName"
Response	changeGroupControlAdded

Usage Example

A Control Alias is removed from the Change Group with the *changeGroupRemoveControl* command. For example:

```
changeGroupControlRemove "Main Gain"\r
```

or

```
cgcr "Main Gain"\r
```

Response

```
changeGroupControlRemoved "Main Gain"\r\n
```

The above response is issued even if the Control Alias named was not in the Change Group.

Possible Error Messages

```
\aBadArgumentCount\r\n
\anotLoggedIn\r\n
\aInvalidChangeGroup "yourBogusGroupName"\r\n
\aUnlistedControl "AliasName"\r\n
\aOverflow\r\n
```

changeGroupGet Command

Command	changeGroupGet
Shortcut	cgg
Arguments	[group]
Availability	when logged in
Purpose	to read the changed values of a Change Group.
Notes	The group name argument is optional. If not included, a Change Group named 'default Change Group' will be used. The number of changes that have occurred is indicated in the first line of the response.
Response	see below

Usage Example

The client issues:

```
changeGroupGet "Balcony Speaker Overload Indicators Change Group"\r
```

or

```
cgg "Balcony Speaker Overload Indicators Change Group"\r
```

All values that have changed will be returned.

Response

RATC2 responds with the number of values that have changed and a value for each change. For example:

```
changeGroupChanges "Balcony Speaker Overload Indicators Change Group"
2\r\n
```

```
valueIs "North 1 OL" on 1.000\r\n
```

```
valueIs "North 3 OL" off 0.000\r\n
```

In this example, there are two Control Aliases in the Change Group that have changed since the last query. If the command is issued again, with no changes to Control Aliases, or if there are no Control Aliases in the Change Group, the response will be:

```
changeGroupChanges "Balcony Speaker Overload Indicators Change Group"
0\r\n
```

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\aOverflow\r\n
```

```
\aNotRunning\r\n
```

```
\aInvalidChangeGroup "yourBogusGroupName"\r\n
```

changeGroupClear Command

Command	changeGroupClear
Shortcut	cgc
Arguments	[group].
Availability	when logged in
Purpose	to destroy a Change Group.
Notes	The group name argument is optional. If not included, a Change Group named 'default Change Group' will be used. It is not necessary to remove the controls from a Change Group before destroying it.
Response	changeGroupCleared

Usage example

The Change Group is cleared of all Control Aliases with the command:

```
changeGroupClear\r
```

or

```
cgc\r
```

Response

```
changeGroupCleared\r\n
```

Possible Error Messages

```
\aBadArgumentCount\r\n
\anotLoggedIn\r\n
\aInvalidChangeGroup "yourBogusGroupName"\r\n
\aOverflow\r\n
```

changeGroupSchedule Command

Command	changeGroupSchedule
Shortcut	cgs
Arguments	[group] seconds
Availability	when logged in
Purpose	To schedule automatic, unsolicited, recurring, changeGroupGets. If any changes have occurred when the periodic timer expires, the server will automatically send a change list, as if changeGroupGet has been called. While this mechanism violates the normal client/server relationship, and is not normally recommended (what if too many changes occur too quickly for the control system?), it may be useful in reducing network traffic if the control system is polling a very large number of servers and is looking for changes that need to be recognized quickly. Otherwise, reasonable real-time programming practices on the control system side make this command unnecessary.
Notes	The group name argument is optional, and if it is not included, a Change Group named default Change Group will be used. The schedule can be cancelled by calling changeGroupSchedule with an argument of zero. Only one Change Group can be scheduled. **Note:** This command is only supported over TCP connections.
Response	Something like this: changeGroupSchedule Balcony Speaker Overload Indicators 5 Subsequently, changeGroupGet responses will be returned when controls in the Change Group have changed.

Usage example

The client issues:

```
changeGroupSchedule "Balcony Speaker Overload Indicators Change Group"
5\r
```

or

```
cgs "Balcony Speaker Overload Indicators Change Group" 5\r
```

Response

RATC2 responds with the Change Group name and the number of changes that were requested.

```
changeGroupSchedule "Balcony Speaker Overload Indicators Change Group"
5\r\n
```

Possible Error Messages

```
\aBadArgumentCount\r\n
```

```
\aOverflow\r\n
```

```
\aInvalidChangeGroup "yourBogusGroupName"\r\n
```

Chapter 4

RATC2 RAW

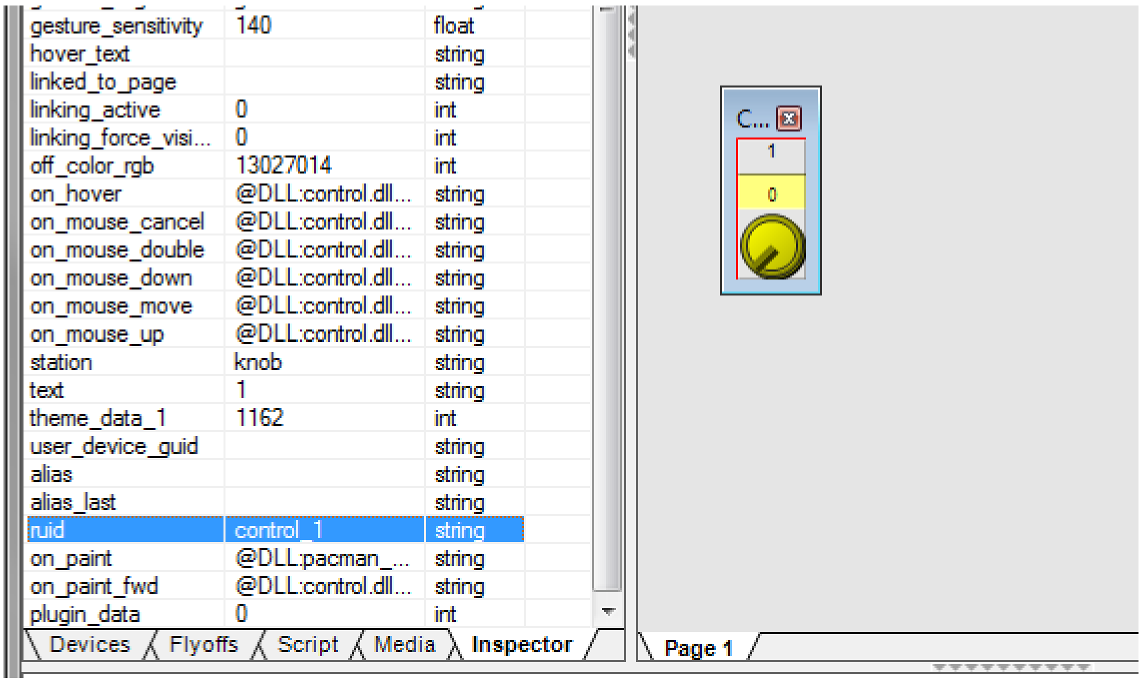
In This Chapter

Introduction

RATC2 RAW is a variant of the RATC2 protocol. It allows you to access controls in a project using control aliases in the same way as RATC2, but the aliases are defined automatically. You do not need to manually specify them for each control in NWare.

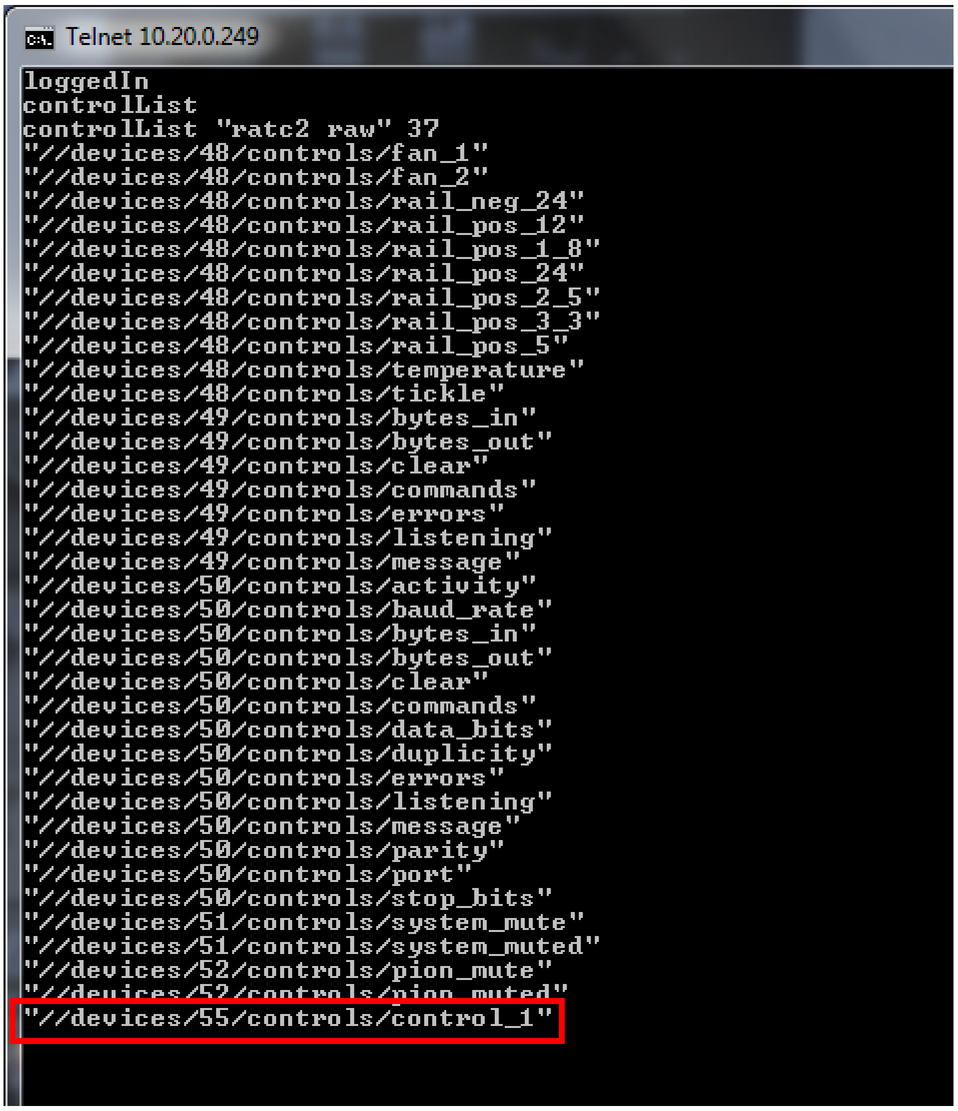
In the example below, the knob control has been automatically assigned a RUID (Relatively Unique ID) of *control_1* by NWare. This name can be used to reference the knob using RATC2 RAW.

Tip: You will need to show the Inspector tab in order to see the RUID value. For more information, see Displaying object values using the Inspector.



gesture_sensitivity	140	float
hover_text		string
linked_to_page		string
linking_active	0	int
linking_force_visi...	0	int
off_color_rgb	13027014	int
on_hover	@DLL:control.dll...	string
on_mouse_cancel	@DLL:control.dll...	string
on_mouse_double	@DLL:control.dll...	string
on_mouse_down	@DLL:control.dll...	string
on_mouse_move	@DLL:control.dll...	string
on_mouse_up	@DLL:control.dll...	string
station	knob	string
text	1	string
theme_data_1	1162	int
user_device_guid		string
alias		string
alias_last		string
ruid	control_1	string
on_paint	@DLL:pacman_...	string
on_paint_fwd	@DLL:control.dll...	string
plugin_data	0	int

When you connect to the MediaMatrix node hosting the project, then type the *controlList* command, you can see the aliases that NWare has defined. These include the alias for the knob control: *control_1*.



```
loggedIn
controlList
controlList "ratc2 raw" 37
"//devices/48/controls/fan_1"
"//devices/48/controls/fan_2"
"//devices/48/controls/rail_neg_24"
"//devices/48/controls/rail_pos_12"
"//devices/48/controls/rail_pos_1_8"
"//devices/48/controls/rail_pos_24"
"//devices/48/controls/rail_pos_2_5"
"//devices/48/controls/rail_pos_3_3"
"//devices/48/controls/rail_pos_5"
"//devices/48/controls/temperature"
"//devices/48/controls/tickle"
"//devices/49/controls/bytes_in"
"//devices/49/controls/bytes_out"
"//devices/49/controls/clear"
"//devices/49/controls/commands"
"//devices/49/controls/errors"
"//devices/49/controls/listening"
"//devices/49/controls/message"
"//devices/50/controls/activity"
"//devices/50/controls/baud_rate"
"//devices/50/controls/bytes_in"
"//devices/50/controls/bytes_out"
"//devices/50/controls/clear"
"//devices/50/controls/commands"
"//devices/50/controls/data_bits"
"//devices/50/controls/duplicity"
"//devices/50/controls/errors"
"//devices/50/controls/listening"
"//devices/50/controls/message"
"//devices/50/controls/parity"
"//devices/50/controls/port"
"//devices/50/controls/stop_bits"
"//devices/51/controls/system_mute"
"//devices/51/controls/system_muted"
"//devices/52/controls/pion_mute"
"//devices/52/controls/pion_muted"
"//devices/55/controls/control_1"
```

Note: If you are working on a large and complex project, we recommend that you use RATC2, rather than RATC2 RAW. As the example shows, RATC2 RAW requires that you discover the name of each control alias after NWare has assigned it. This process may need to be repeated many times. If you use RATC2, you can use the Expression Labeler to assign aliases to multiple controls simultaneously. This can save a lot of time.

Chapter 5

PASHA

In This Chapter

Introduction

PASHA, the MediaMatrix Serial Handling Adapter, is a remote control protocol that provides external serial control and read-back of any of the controls appearing in an NWare project.

Notes:

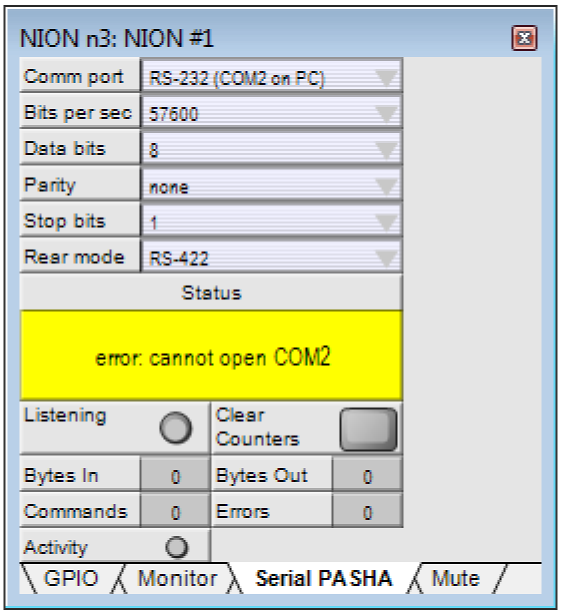
- By default, console mode on the NION RS-232 serial port is disabled. In order to use PASHA on this port, it must remain disabled. For more information on configuring the port, refer to the section *Specifying the function of the RS-232 serial port* in the *NION Hardware Manual*.
- You cannot specify PASHA for one serial port on a NION and RATC for the other serial port. However, you can specify different variants of PASHA on the two serial ports; for example, PASHA/Legacy and PASHA/X-Control.

Allowing PASHA to be used on a MediaMatrix node

1. Right-click the NioNode, nControl node or nTouch 180 node, and then click **Device Properties**.
2. In the **Serial Control Protocol** list, click a PASHA option.

PASHA/PageMatrix	Protocol used with the PageMatrix Command Center. It supports a four character control ID sent with the (T) trigger command.
PASHA/XControl	Provides basic S (Set) PASHA functionality with X-Net2-style trigger commands.
PASHA/Legacy	Designed for projects that contain legacy MediaMatrix nodes that are controlled by external programs using the classic PASHA protocol, as implemented on MainFrames and MiniFrames. When you want to replace the legacy nodes with NioNodes, select the *PASHA/Legacy* option and you will not need to update your external program code. This option ensures all hex values are returned from nodes in lower case instead of upper case to match the original MediaMatrix PASHA protocol.

3. Deploy the project, and then specify settings on the **Serial PASHA** tab to match your control system. For example:



NION n3: NION #1

Comm port	RS-232 (COM2 on PC)
Bits per sec	57600
Data bits	8
Parity	none
Stop bits	1
Rear mode	RS-422

Status

error: cannot open COM2

Listening		Clear Counters	
Bytes In	0	Bytes Out	0
Commands	0	Errors	0
Activity			

Testing and debugging

Using MediaMatrix hardware

You can use a terminal emulator program to manually send ASCII PASHA commands to a MediaMatrix node via one of the available serial ports.

Note: If you are using a NION RS-232 port, console mode must be disabled. See the section *Specifying the function of the RS-232 serial port* in the *NION Hardware Manual*.

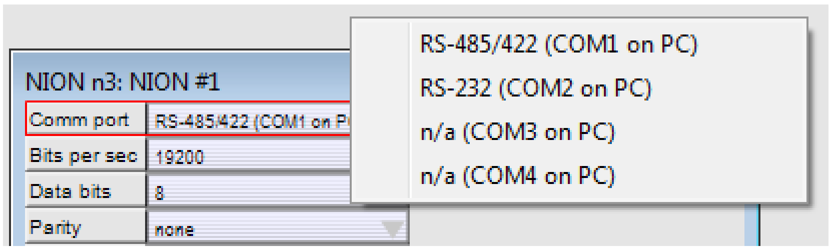
To use MediaMatrix hardware

1. Connect a serial cable between the PC serial port and the serial port on the MediaMatrix node.
2. Start a terminal emulator program, then specify connection settings to match those already specified on the MediaMatrix node. For example (to match the settings in the earlier example):
   - baud rate 57600
   - 8 bit data
   - no parity
   - 1 stop bit
   - no flow control.
3. Enable the echo feature in the terminal emulator program if it is switched off.

   In Hyper Terminal, you can do this by clicking **ASCII Setup**, and then selecting the **Echo typed characters locally** check box.

Using a PC

The serial command protocol used by PASHA is human-readable ASCII, so you can test and debug PASHA control of a specific project using just a Windows-based PC with a spare serial port – MediaMatrix hardware is not required. When emulating a MediaMatrix node using NWare on a PC, the PC serial ports can simulate the PASHA behavior on the node.


NION n3: NION #1

Comm port	RS-485/422 (COM1 on P
Bits per sec	19200
Data bits	8
Parity	none

Note: If you want to use RS-485/422 with your PC, it must be fitted with an RS-485/422 serial interface card on COM1, or you need to use an interface converter device plugged into the COM-1 RS-232 port.

PASHA user IDs and control aliases

In order to refer to NWare controls using the PASHA protocol, you need to assign a *user ID* to each control. User IDs are equivalent to *control aliases* in the RATC protocol, except they are limited to three alphanumeric characters.

Notes:

- Any alphanumeric character may be used, but the protocol is case-sensitive, so *aaa* is different from *AAA*.
- Only aliases that are exactly three characters long can be used.

For information on assigning user IDs / control aliases, see *Controlling NWare controls* (on page 11).

Message protocol

PASHA messages are ASCII (text) strings. While they contain hexadecimal numbers, the hex numbers are represented by ASCII (text).

Each message begins with a message-type character and ends (with the exception of the NAK message) with an end-of-message period character (.). Each message to PASHA results in a response message from PASHA. PASHA never sends a message except in response to an external message.

In general, the protocol supports setting and getting control values. The alphanumeric characters used to identify a control are set as a control alias on the properties of the control in NWare. Control values are specified with a 2 digit hexadecimal number, in the range 00 through FF.

Getting and setting control values

Controls are set to a position using the cSETVALUE 'S' command. You can determine the current setting of a control using the cGETVALUE 'G' command. The 256 values that a control can be assigned correspond to 256 equidistant positions of a control in an NWare project.

Note: It is not required or desirable to put a carriage return after the end-of-message (EOM) character (.). PASHA will accept the command when it receives the EOM character. Shortly thereafter it will return the value that the control has been set to.

To check what a control is set to, you can use the cGETVALUE 'G' command. For example, to check the setting of a channel 1 parametric EQ center frequency control with an control alias of *tes*, you send:

```
Gtes.
```

Since you are requesting a control value, you do not need to send one. If the control was set to the midpoint of its rotation you will get a response:

```
Vtes80.
```

If you send a command using a user ID (alias) that does not exist in the project, PASHA will respond with a cUNLISTEDUID 'U' message. For example if there was no alias *eje* in the project, and you sent:

```
Seje48.
```

the response would be:

```
U.
```

Message protocol (boolean control)

If you want to control a button, such as a mute button, any value between 00 and 7F sent to the control will result in a return value of 00. Any value between 80 and FF will result in a return value of FF.

For example, if you send the following command to mute channel 2:

```
S2oM3E.
```

the channel mute button will be activated. Since the switch can only be on or off, it will be set to off, and return:

```
V2oM00.
```

Message protocol (gain control)

The gain controls on most devices are non-linear. For those controls, tables are included at the end of this section that map linear control values to dBs of gain, one for a gain control with range of -100dB to +18dB, another for a gain control with a range of -100dB to 0dB. Also included are tables that map *control values to router channel selections* (on page 62) for various sized routers.

For example, to set the channel 2 input gain for a control with an ID of *2iG* to fully counter-clockwise (value -100), we could use:

```
S2iG00.
```

This uses the cSETVALUE command abbreviated to S. The end-of-message character is a period (.).

And the response would be:

```
V2iG00.
```

C-like message definition

This section defines the message protocol using *C-like* declarations of constants and structures. The convention here is that words beginning with *c*, such as *cSETVALUE* and *cGETVALUE*, are character constants. Words beginning with *f*, such as *fUid* and *fVal*, are message field structures. Words beginning with *m*, such as *mSetValue* and *mGetValue*, are message structures.

```
//---- Message constants

const char cSETVALUE = 'S'; // an fType value

const char cGETVALUE = 'G'; // an fType value

const char cVALUE = 'V'; // an fType value

const char cNOTREADY = 'R'; // an fType value

const char cUNLISTEDUID = 'U'; // an fType value

const char cFAIL = 'X'; // an fType value

const char cNAK = '?'; // an fType value

const char cEOM = '.'; // an fEom value


//---- Message fields
/* fType: Message Type. 1 character, denotes the message type. */

struct fType {

  char data;

};

/* fUid: User ID. 3 ASCII characters, specifies one of the User IDs entered
as a control alias in the project. */

struct fUid {

  char data[3];

};


/* fVal: Control Value. 2 ASCII hexadecimal digits, specifies a control
value between 0 and 255. */

struct fVal {

  char data[2];

};


/* fEom: End Of Message. 1 character, appears at the end of every message,
excepting mNak. */

struct fEom {
  char data = cEOM;

};


//---- Messages

/* mNak: Negative Acknowledge. Response sent to client upon receipt of
unintelligible data. This could be due to a communications error or to
data out of order. An mNak is not necessarily sent for every byte of bad
data. */

struct mNak {

  fType = cNAK;

};


/* mSetValue: Set Control Value. Request sent by a client to set the value
of a control identified by a control alias. */

struct mSetValue {

  fType type = cSETVALUE;

  fUid id;

  fVal val;

   fEom eom;

};


/* mGetValue: Get Control Value. Request sent by a client requesting the
value of a control identified by a control alias. */

struct mGetValue {

  fType type = cGETVALUE;

  fUid id;

  fEom eom;

}};


/* mValue: Control Value. Response sent to the client acknowledging an
mGetValue or mSetValue. Note that it is possible, and normal in some
cases, that the val field will not match the val that was sent in the
mSetValue. */

struct mValue {

  fType type = cVALUE;

  fUid id;

  fVal val;

  fEom eom;

};
```

```
//* mUnlistedUid: Unlisted User ID error. Response indicating the fUid
specified in the mSetValue or mGetValue does not match any control aliases
in the currently running project. */

struct mUnlistedUid {

  fType type = cUNLISTEDUID;

  fUid id;

  fEom eom;

};

/* mNotReady: Not Ready. This means that there is no project currently
compiled and running in MediaMatrix. */

struct mNotReady {

  fType type = cNOTREADY;

  fEom eom;

};


/* mFail: Something Has Failed. This is sent in response to serial port
errors, communication time-outs, and other internal errors not covered
directly. */

struct mFail {

  fType type = cFAIL;

  fEom eom;

};
```

Message structures quick chart

Message Name	**Message Fields**				
	Type	**Alphanumeric UID**	**Hex value**	**End of message**	**Style**
Set value	S	XXX	XX	.	Request
Get value	G	XXX		.	Request
Value	V	XXX	XX	.	Response
Not ready	R			.	Response
Unlisted UID	U	XXX		.	Response
Fail	X			.	Response
Nak	?				Response

So, for example, setting a value with a user ID value *tes* would look like this *Stes44.*, the result will look like this *Stes44.Vtes44.*

Serial Control Value to Device Control Value Tables

This section shows some example mappings of serial control values to device control values. An NWare knob control, for example, could have a range of 0 to 1, but the values returned by the PASHA G (get) command would be in the hexadecimal range 0 - ff.

Note: The tables in this section do not show *0x* at the start of the hex values, as it is not returned as part of the PASHA message.

Control value to dB's of gain table: -100dB to +18.0dB type control

00: -100dB	01: -99.5dB	02: -99.1dB	03: -98.6dB
04: -98.1dB	05: -97.7dB	06: -97.2dB	07: -96.8dB
08: -96.3dB	09: -95.8dB	0a: -95.4dB	0b: -94.9dB
0c: -94.4dB	0d: -94.0dB	0e: -93.5dB	0f: -93.1dB
10: -92.6dB	11: -92.1dB	12: -91.7dB	13: -91.2dB
14: -90.7dB	15: -90.3dB	16: -89.8dB	17: -89.4dB
18: -89.9dB	19: -88.4dB	1a: -88.0dB	1b: -87.5dB
1c: -87.0dB	1d: -86.6dB	1e: -86.1dB	1f: -85.7dB
20: -85.2dB	21: -84.7dB	22: -84.3dB	23: -83.8dB
24: -83.3dB	25: -82.9dB	26: -82.4dB	27: -82.0dB
28: -81.5dB	29: -81.0dB	2a: -80.6dB	2b: -80.1dB
2c: -79.6dB	2d: -79.2dB	2e: -78.7dB	2f: -78.3dB
30: -77.8dB	31: -77.3dB	32: -76.9dB	33: -76.4dB
34: -75.9dB	35: -75.5dB	36: -75.0dB	37: -74.5dB
38: -74.1dB	39: -73.6dB	3a: -73.2dB	3b: -72.7dB
3c: -72.2dB	3d: -71.8dB	3e: -71.3dB	3f: -70.8dB
40: -70.4dB	41: -69.9dB	42: -69.5dB	43: 69.0-dB
44: -68.5dB	45: -68.1dB	46: -67.6dB	47: -67.1dB

48: -66.7dB	49: -66.2dB	4a: -65.8dB	4b: -65.3dB
4c: -64.8dB	4d: -64.4dB	4e: -63.9dB	4f: -63.4dB
50: -63.0dB	51: -62.5dB	52: -62.1dB	53: -61.6dB
54: -61.1dB	55: -60.7dB	56: -60.2dB	57: -59.7dB
58: -59.3dB	59: -58.8dB	5a: -58.4dB	5b: -57.9dB
5c: -57.4dB	5d: -57.0dB	5e: -56.5dB	5f: -52.3dB
60: -55.6B	61: -55.1dB	62: -54.7dB	63: -54.2dB
64: -53.7dB	65: -53.3dB	66: -52.8dB	67: -52.5dB
68: -51.9dB	69: -51.4dB	6a: -50.9dB	6b: -50.5dB
6c: -50.0dB	6d: -49.6dB	6e: -49.1dB	6f: -48.6dB
70: -48.2dB	71: -47.7dB	72: -47.2dB	73: -46.8dB
74: -46.3dB	75: -45.9dB	76: -45.4dB	77: -44.9dB
78: -44.5dB	79: -44.0dB	7a: -43.5dB	7b: -43.1dB
7c: -42.6dB	7d: -42.2dB	7e: -41.7dB	7f: -41.2dB
80: -40.8dB	81: -40.3dB	82: -39.8dB	83: -39.4dB
84: -38.9dB	85: -38.5dB	86: -38.0dB	87: -37.5dB
88: -37.1dB	89: -36.6dB	8a: -36.1dB	8b: -35.7dB
8c: -35.2dB	8d: -34.8dB	8e: -34.3dB	8f: -33.8dB
90: -33.4dB	91: -32.9dB	92: -32.4dB	93: -32.0dB
94: -31.5dB	95: -31.1dB	96: -30.6dB	97: -30.1dB
98: -29.7dB	99: -29.2dB	9a: -28.7dB	9b: -28.3dB
9c: -27.8dB	9d: -27.3dB	9e: -26.9dB	9f: -26.4dB
a0: -26.0dB	a1: -25.5dB	a2: -25.0dB	a3: -24.6dB
a4: -24.1dB	a5: -23.6dB	a6: -23.2dB	a7: -22.7dB
a8: -22.3dB	a9: -21.8dB	aa: -21.3dB	ab: -20.9dB
ac: -20.4dB	ad: -19.9dB	ae: -19.5dB	af: -19.0dB

b0: -18.6dB	b1: -18.1dB	b2: -17.6dB	b3: -17.2dB
b4: -16.7dB	b5: -16.2dB	b6: -15.8dB	b7: -15.3dB
b8: -14.9dB	b9: -14.4dB	ba: -13.9dB	bb: -13.5dB
bc: -13.0dB	bd: -12.5dB	be: -12.1dB	bf: -11.6dB
c0: -11.2dB	c1: -10.7dB	c2: -10.2dB	c3: -9.76dB
c4: -9.30dB	c5: -8.84dB	c6: -8.38dB	c7: -7.91dB
c8: -7.45dB	c9: -6.99dB	ca: -6.53dB	cb: -6.06dB
cc: -5.60dB	cd: -5.14dB	ce: -4.67dB	cf: -4.21dB
d0: -3.75dB	d1: -3.29dB	d2: -2.82dB	d3: -2.36dB
d4: -1.90dB	d5: -1.44dB	d6: -0.973dB	d7: -0.510dB
d8: -0.047dB	d9: +0.416dB	da: +0.878dB	db: +1.34dB
dc: +1.80dB	dd: +2.27dB	de: +2.73dB	df: +3.19dB
e0: +3.65dB	e1: +4.12dB	e2: +4.58dB	e3: +5.04dB
e4: +5.51dB	e5: +5.97dB	e6: +6.43dB	e7: +6.89dB
e8: +7.36dB	e9: +7.82dB	ea: +8.28dB	eb: +8.75dB
ec: +9.21dB	ed: +9.67dB	ee: +10.1dB	ef: +10.6dB
f0: +11.1dB	f1: +11.5dB	f2: +12.0dB	f3: +12.4dB
f4: +12.9dB	f5: +13.4dB	f6: +13.8dB	f7: +14.3dB
f8: +14.8dB	f9: +15.2dB	fa: +15.7dB	fb: +16.1dB
fc: +16.6dB	fd: +17.1dB	fe: +17.5dB	ff: +18.0dB

Control value to dB's of gain table: -100dB to +0.00dB type control

00: -100dB	01: -99.6dB	02: -99.2dB	03: -98.8dB
04: -98.4dB	05: -98.0dB	06: -97.6dB	07: -97.3dB
08: -96.9dB	09: -96.5dB	0a: -96.1dB	0b: -95.7dB

0c: -95.3dB	0d: -94.9dB	0e: -94.5dB	0f: -94.1dB
10: -93.7dB	11: -93.3dB	12: -92.9dB	13: -92.5dB
14: -92.2dB	15: -91.8dB	16: -91.4dB	17: -91.0dB
18: -90.6dB	19: -90.2dB	1a: -89.8dB	1b: -89.4dB
1c: -89.0dB	1d: -88.6dB	1e: -88.2dB	1f: -87.8dB
20: -87.5dB	21: -87.1dB	22: -86.7dB	23: -86.3dB
24: -85.9dB	25: -85.5dB	26: -85.1dB	27: -84.7dB
28: -84.3dB	29: -83.9dB	2a: -83.5dB	2b: -83.1dB
2c: -82.7dB	2d: -82.4dB	2e: -82.0dB	2f: -81.6dB
30: -81.2dB	31: -80.8dB	32: -80.4dB	33: -80.0dB
34: -79.6dB	35: -79.2dB	36: -78.8dB	37: -78.4dB
38: -78.0dB	39: -77.6dB	3a: -77.3dB	3b: -76.9dB
3c: -76.5dB	3d: -76.1dB	3e: -75.7dB	3f: -75.3dB
40: -74.9dB	41: -74.5dB	42: -74.1dB	43: -73.7dB
44: -73.3dB	45: -72.9dB	46: -72.5dB	47: -72.2dB
48: -71.8dB	49: -71.4dB	4a: -71.0dB	4b: -70.6dB
4c: -70.2dB	4d: -69.8dB	4e: -69.4dB	4f: -69.0dB
50: -68.6dB	51: -68.2dB	52: -67.8dB	53: -67.5dB
54: -67.1dB	55: -66.7dB	56: -66.3dB	57: -65.9dB
58: -65.5dB	59: -65.1dB	5a: -64.7dB	5b: -64.3dB
5c: -63.9dB	5d: -63.5dB	5e: -63.1dB	5f: -62.7dB
60: -62.4dB	61: -62.0dB	62: -61.6dB	63: -61.2dB
64: -60.8dB	65: -60.4dB	66: -60.0dB	67: -59.6dB
68: -59.2dB	69: -58.8dB	6a: -58.4dB	6b: -58.0dB
6c: -57.6dB	6d: -57.3dB	6e: -56.9dB	6f: -56.5dB
70: -56.1dB	71: -55.7dB	72: -55.3dB	73: -54.9dB

74: -54.5dB	75: -54.1dB	76: -53.7dB	77: -53.3dB
78: -52.9dB	79: -52.5dB	7a: -52.2dB	7b: -51.8dB
7c: -51.4dB	7d: -51.0dB	7e: -50.6dB	7f: -50.2dB
80: -49.8dB	81: -49.4dB	82: -49.0dB	83: -48.6dB
84: -48.2dB	85: -47.8dB	86: -47.5dB	87: -47.1dB
88: -46.7dB	89: -46.3dB	8a: -45.9dB	8b: -45.5dB
8c:-45.1dB	8d: -44.7dB	8e: -44.3dB	8f: -43.9dB
90: -43.5dB	91: -43.1dB	92: -42.7dB	93: -42.4dB
94: -42.0dB	95: -41.6dB	96: -41.2dB	97: -40.8dB
98: -40.4dB	99: -40.0dB	9a: -39.6dB	9b: -39.2dB
9c: -38.8dB	9d: -38.4dB	9e: -38.0dB	9f: -37.6dB
a0: -37.3dB	a1: -36.9dB	a2: -36.5dB	a3: -36.1dB
a4: -35.7dB	a5: -35.3dB	a6: -34.9dB	a7: -34.5dB
a8: -34.1dB	a9: -33.7dB	aa: -33.3dB	ab: -32.9dB
ac: -32.5dB	ad: -32.2dB	ae: -31.8dB	af: -31.4dB
b0: -31.0dB	b1: -30.6dB	b2: -30.2dB	b3: -29.8dB
b4: -29.4dB	b5: -29.0dB	b6: -28.6dB	b7: -28.2dB
b8: -27.8dB	b9: -27.5dB	ba: -27.1dB	bb: -27.6dB
bc: -26.3dB	bd: -25.9dB	be: -25.5dB	bf: -25.1dB
c0: -24.7dB	c1: -24.3dB	c2: -23.9dB	c3: -23.5dB
c4: -23.1dB	c5: -22.7dB	c6: -22.4dB	c7: -22.0dB
c8: -21.6dB	c9: -21.2dB	ca: -20.8dB	cb: -20.4dB
cc: -20.0dB	cd: -19.6dB	ce: -19.2dB	cf: -18.8dB
d0: -18.4dB	d1: -18.0dB	d2: -17.6dB	d3: -17.3dB
d4: -16.9dB	d5: -16.5dB	d6: -16.1dB	d7: -15.7dB
d8: -15.3dB	d9: -14.9dB	da: -14.5dB	db: -14.1dB

dc: -13.7dB	dd: -13.3dB	de: -12.9dB	df: -12.5dB
e0: -12.0dB	e1: -11.8dB	e2: -11.4dB	e3: -11.0dB
e4: -10.6dB	e5: -10.2dB	e6: -9.80dB	e7: -9.41dB
e8: -9.02dB	e9: -8.63dB	ea: -8.24dB	eb: -7.84dB
ec: -7.45dB	ed: -7.06dB	ee: -6.67dB	ef: -6.27dB
f0: -5.88dB	f1: -5.49dB	f2: -5.10dB	f3: -4.71dB
f4: -4.31dB	f5: -3.92dB	f6: -3.53dB	f7: -3.14dB
f8: -2.75dB	f9: -2.35dB	fa: -1.96dB	fb: -1.57dB
fc: -1.18dB	fd: -0.784dB	fe: -0.392dB	ff: +0.00dB

Control value to router input selection table: 1x1 router (on/off switch)

Input (with off state)	Value Range
Off	00 - 7f
1	80 - ff

Tip: A 1x1 router device will consume fewer DSP resources than a single channel gain device. Therefore, if you require only the mute functionality from one of these devices, we recommend that you use a router device.

Control value to Router input selection table: 2x1 router

Input (with off state)	Value Range
Off	00 -3f
1	40 - bf
2	C0 - ff

Input (without off state)	Value Range
1	00 - 7f
2	80 - ff

Control value to Router input selection table: 3x1 router

Input (with off state)	Value Range
Off	00 - 2a
1	2B - 7f
2	80 - d4
3	d5 - ff

Input (without off state)	Value Range
1	00 - 3f
2	40 - bf
3	c0 - ff

Control value to Router input selection table: 4x1 Router

Input (with off state)	Value Range
Off	00 - 0x1f
1	20 - 0x5f
2	60 - 0x9f
3	A0 - 0xdf
4	E0 - 0xff

Input (with no off state)	**Value Range**
1	00 - 2a
2	2b - 7f
3	80 - d4
4	D5 - ff

Control value to Router input selection table: 8x1 Router

Input (with off state)	**Value Range**
Off	00 - 0f
1	10 - 2f
2	30 - 4f
3	50 - 6f
4	70 - 8f
5	90 - af
6	b0 - cf
7	d0 - ef
8	f0 - ff

Input (with no off state)	**Value Range**
1	00 - 12
2	13 - 36
3	37 - 5b
4	5c - 7f
5	80 - a3
6	a4 - c8

Input (with no off state)	**Value Range**
7	c9 - ec
8	ed - ff

Chapter 6

NioNode SNMP control

In This Chapter

Introduction

NioNodes support Simple Network Management Protocol (SNMP). This allows a third party system to monitor NioNodes in an NWare project and get and set control values.

Note: Currently, a third party system cannot use SNMP to interact with nControl and nTouch 180 projects. However, a project hosted by one of these nodes can monitor other devices on the network using SNMP. For more information, see *SNMP Components* in the *NWare Device Reference*.

SNMP OIDs and MIB files

SNMP uses a system of object identifiers (OIDs) or SNMP addresses to provide unique locations for items of data. In each case, you can read a value or replace it with a new one.

NWare also includes a MIB (Management Information Base) file. The NioNode SNMP MIB file is a standardized plain-text format file that defines the available OIDs. This is an *uncompiled* MIB file.

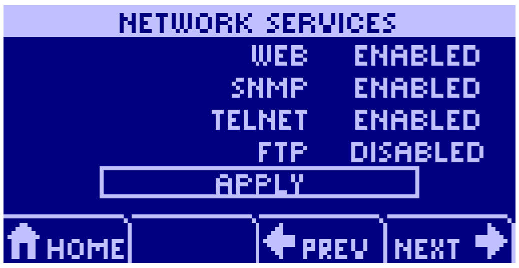
The NioNode MIB file allows you to replace or add understandable names to guide you through the otherwise all-numeric OIDs.   The MIB file was designed alongside the SNMP agent that runs on the NioNode hardware.

Enabling SNMP on the NioNode

The SNMP network service must be enabled on the NioNode before you can use SNMP. This can be done using the front panel or the web interface.

⏩ *To enable SNMP using the front panel*

1. From the main menu, select **CONFIG** to display the first configuration page, **LAN CONFIG**, then select **NEXT** repeatedly until the **NETWORK SERVICES** page is displayed.



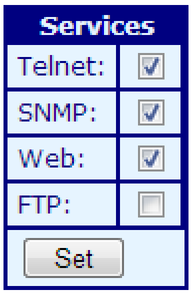
NETWORK SERVICES
WEB ENABLED
SNMP ENABLED
TELNET ENABLED
FTP DISABLED

2. Use the wheel and wheel push button to enable the SNMP service.
3. Select **APPLY** and press the wheel button to complete the adjustment.

⏩ *To enable SNMP using the web interface*

1. Navigate to the **Network** screen.

2. Under Services, select the check box next to **SNMP**.



Services	
Telnet:	✓
SNMP:	✓
Web:	✓
FTP:	☐

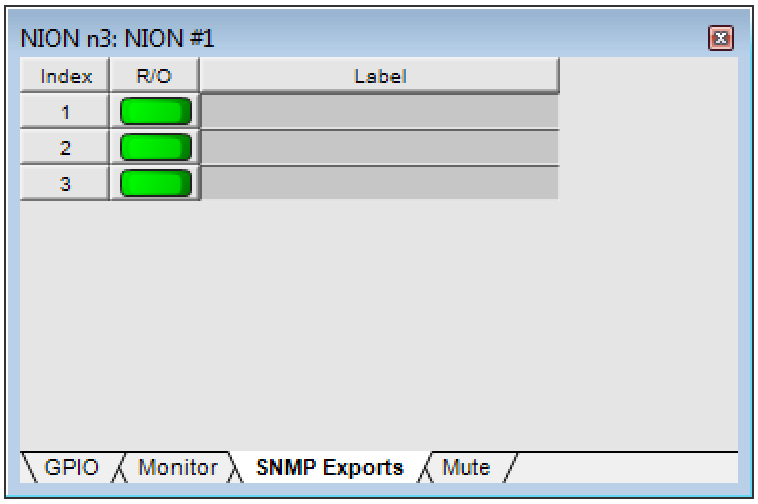
Setting up the NWare SNMP project

Up to 255 NWare controls can be accessed in a project using SNMP. You need to manually specify the number of controls you want to access. You can optionally assign them labels. Each control is then exported as an OID.

To set up the NWare SNMP project

1. Expand the **Hardware** folder in the device tree, then drag a NioNode over to the design page.
2. In the **Number of SNMP exported controls** box on the device properties, specify the number of controls (knobs, faders, LEDs etc.) you would like to access using SNMP.
3. Click **OK**.

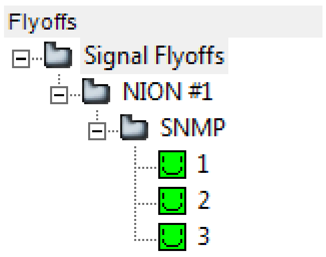
   An SNMP Exports tab will now be available on the NioNode control surface. This allows you to toggle controls between read-only and read/write and specify labels that will be available in your SNMP program.



NION n3: NION #1

Index	R/O	Label
1	🟩	
2	🟩	
3	🟩	

SNMP flyoffs will be added to the Flyoffs tab. You can wire NWare controls to the flyoffs so the control values are automatically passed to the SNMP program, and the SNMP program can send values back to the controls.

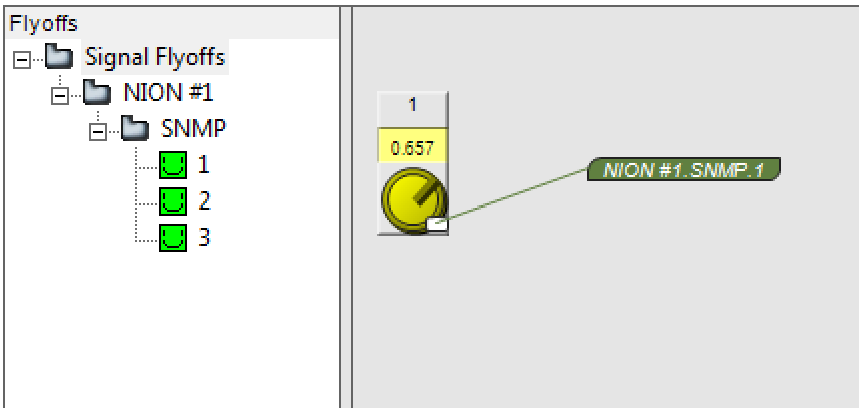


Wiring NWare controls to SNMP flyoffs

In order to pass values to the SNMP program and receive values from the program, you need to wire NWare controls to SNMP flyoffs.

To wire NWare controls to SNMP flyoffs

1. Click the **Flyoffs** tab.
2. Expand the tree to show the flyoffs under **SNMP**.
3. Drag a flyoff over to the design page.
4. Create a master wiring node on the control you want to control using SNMP.
5. Wire the control to the SNMP flyoff.

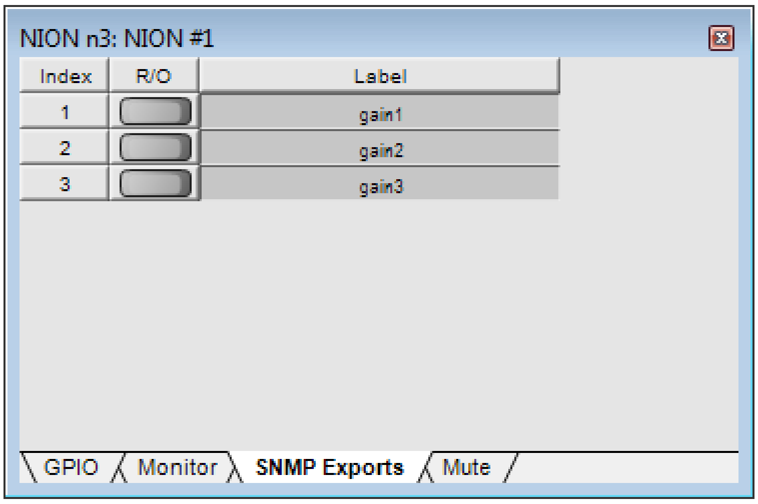


6. Repeat steps 3-5 for the other controls you want to use in the SNMP program.

Labeling controls and setting read/write attributes

1. Open the NioNode block, and then click the **SNMP Exports** tab.
2. In the **Label** box next to each control, type a label that you want to use to refer to the control.
3. If you want to allow the control value to be set, click the **R/O** (read-only) button to switch it *off*.

If you want to prevent the control value from being set, click the **R/O** (read-only) button to switch it *on (green)*.



Index	R/O	Label
1		gain1
2		gain2
3		gain3

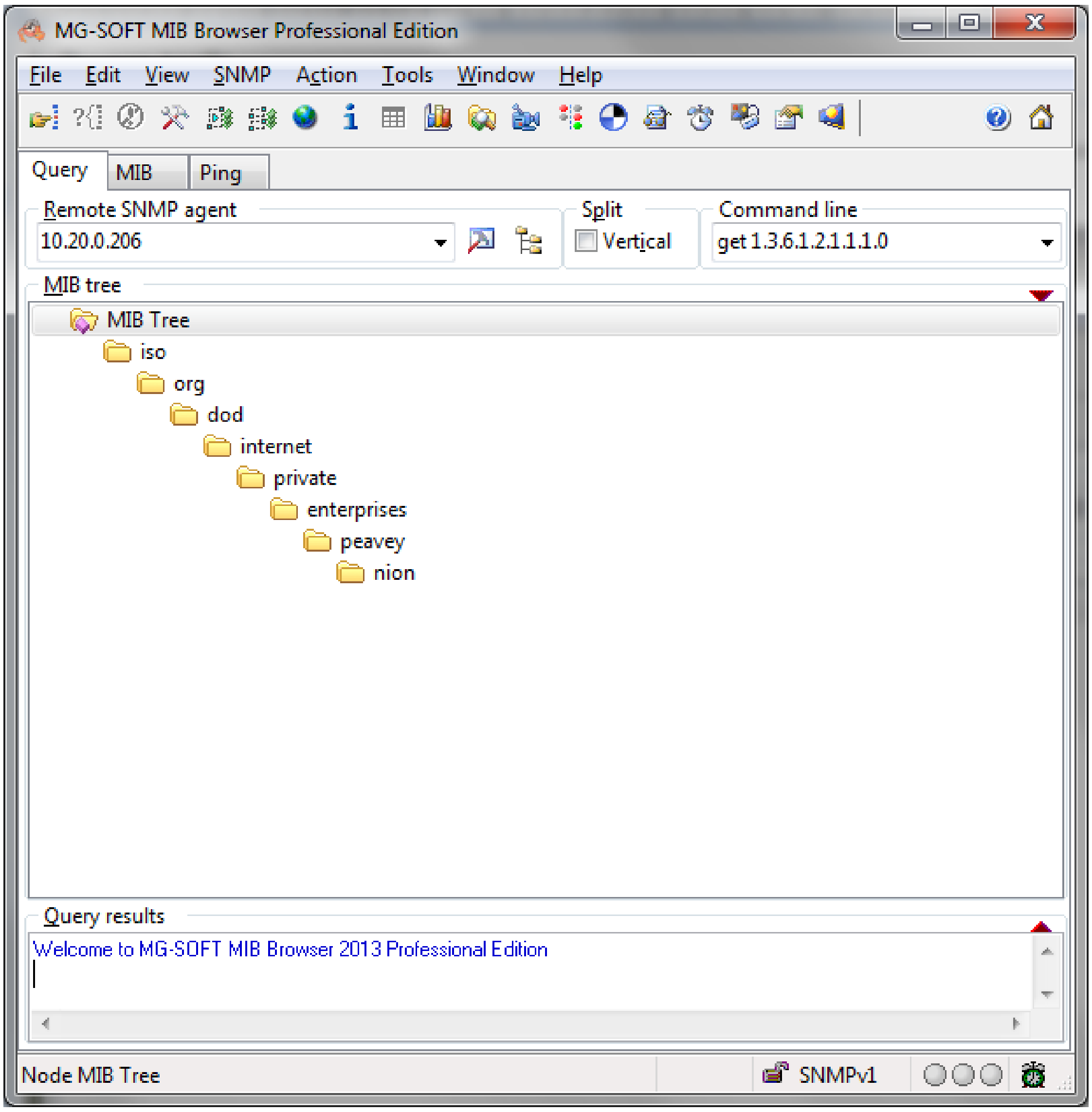
Accessing exported NWare controls

Each exported NWare SNMP control has four variables associated with it: index, label, type and value. Each variable can be accessed using a base OID to which an SNMP control index (1-255) is appended. The SNMP control index is represented by x in the table below.

Index	The index of the exported control. Data type is integer. 1.3.6.1.4.1.24603.1.1.6.4.1.1.x	
Label	A label for the control. This is blank, by default. It can be set on the SNMP Exports tab of the NioNode device in NWare. Data type is string. 1.3.6.1.4.1.24603.1.1.6.4.1.2.x	
Type	The read/write status of the control value. Data type is integer. 1.3.6.1.4.1.24603.1.1.6.4.1.3.x	
	1	Read-only
	2	Read/write
Value	The control value as a string. When the value is read, it will be returned along with the unit of measurement, for example, 100dB or 300Hz. However, When the value is written, you do not need to specify the unit of measurement. Data type is string. 1.3.6.1.4.1.24603.1.1.6.4.1.4.x	

Using a software tool to get and set values via SNMP

You can use a software tool, such as *MIB Browser* (*http://www.mg-soft.com/mgMibBrowserPE.html*) by MG-Soft, to get and set values on a NioNode using SNMP. You can also load the MIB file, compile it, and then view the available settings on the NioNode in a tree view.


MG-SOFT MIB Browser Professional Edition
File Edit View SNMP Action Tools Window Help
Query MIB Ping

Remote SNMP agent: 10.20.0.206
Split: Vertical
Command line: get 1.3.6.1.2.1.1.1.0

MIB tree
- MIB Tree
  - iso
    - org
      - dod
        - internet
          - private
            - enterprises
              - peavey
                - nion

Query results
Welcome to MG-SOFT MIB Browser 2013 Professional Edition

Node MIB Tree | SNMPv1

More advanced products like Ipswitch's *What'sUp* (*http://www.whatsupgold.com/products/*) can perform custom actions, send emails, create web-based reports on network conditions, uptime and anything else within your NION project. Higher end products, such as Hewlett Packard's OpenView, can do this and more.

Tip: All the screenshots in this section are from MIB Browser.

Installing the SNMP program and compiling the MIB file

NWare includes a NION MIB file that is required in order to use SNMP with a NioNode. However, this file is uncompiled and needs to be compiled before you can use it. It can only then be used with an SNMP program, like MIB Browser.

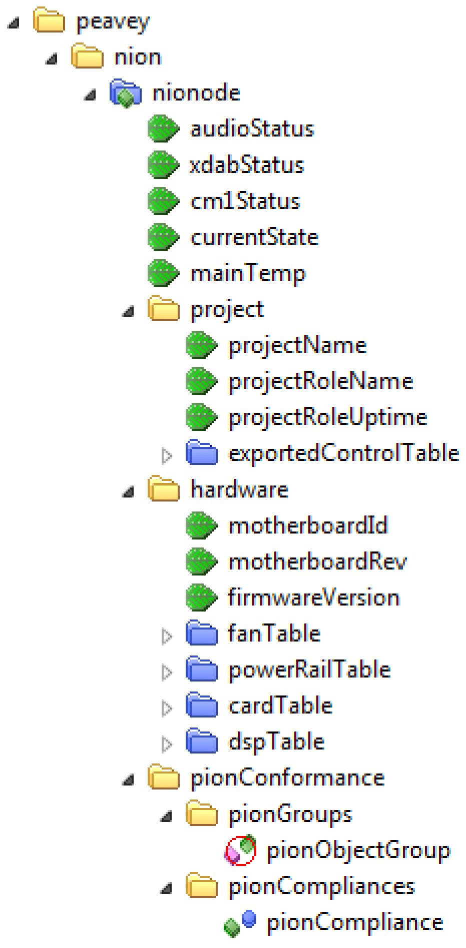
⏩ *Installing the SNMP program and compiling the MIB file*

1. Download the SNMP program and follow the instructions to complete the installation.

   You can download MG-SOFT MIB Browser *here* (*http://www.mg-soft.com/mgMibBrowserPE.html*).
2. Open the MIB compiler program.
3. Open the uncompiled NION MIB file, PEAVEY-NION-NIONODE-MIB-V2.my in the *<NWare install>\plugins\pion\doc* folder.
4. Compile the file to create a *.MIB* file.

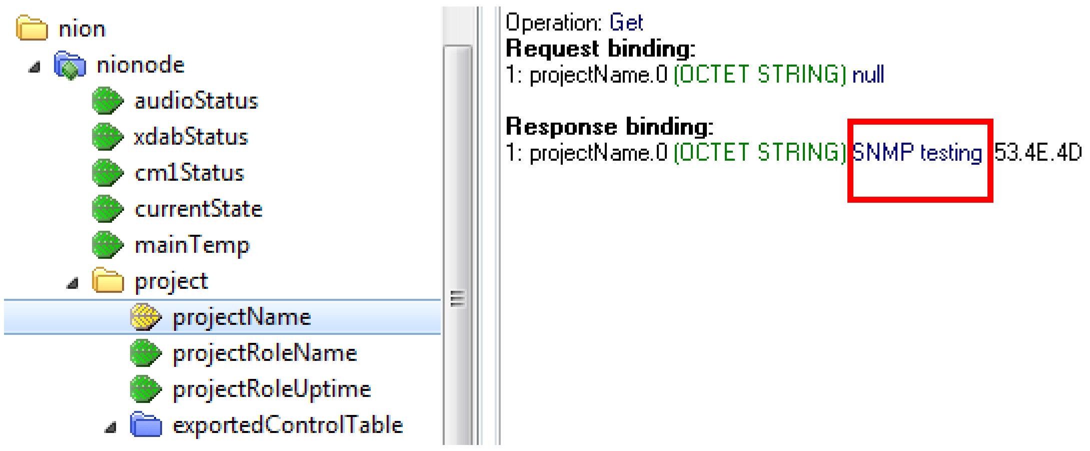
Connecting the SNMP program to the NioNode

1. Start the SNMP program and load the NION MIB file.
2. Connect to the NION using its IP address.
3. Expand the MIB tree to show the nodes under the **nionode** part of the tree.



```
peavey
  nion
    nionode
      audioStatus
      xdabStatus
      cm1Status
      currentState
      mainTemp
      project
        projectName
        projectRoleName
        projectRoleUptime
        exportedControlTable
      hardware
        motherboardId
        motherboardRev
        firmwareVersion
        fanTable
        powerRailTable
        cardTable
        dspTable
      pionConformance
        pionGroups
          pionObjectGroup
        pionCompliances
          pionCompliance
```

Here is an example *projectName* value (SNMP testing) read from the NioNode using the *get* feature.



Operation: Get
Request binding:
1: projectName.0 (OCTET STRING) null

Response binding:
1: projectName.0 (OCTET STRING) SNMP testing 53.4E.4D

Getting and setting NWare control values

1. Expand the MIB tree to show the nodes under the **nionode** part of the tree.
2. Expand the **project/exportedControlTable/exportedControlEntry** part of the tree.
3. If you want to get a list of NWare controls by label, get the values for **controlLabel**.
   If you want to view the values of all NWare controls, get the values for **controlValue**.
   If you want to write a value to an NWare control:
   a) Right-click the **controlValue** node and click the **Set** option.
   b) Double-click the control details in the list and type a value when prompted.

MediaMatrix®

A Division of Peavey Electronics Corp.

5022 Hartley Peavey Drive, Meridian Mississippi, 39305, USA

Phone: 866.662.8750

http://www.peaveycommercialaudio.com/products.cfm/mm/

Features & Specifications subject to change without notice

80307496