# SPN32i Command Set v1.5

# Table of Contents

# SPN32i General Commands

| | |
|---|---|
| baud | Serial port baud rate |
| desc | Device description string |
| fanstat | Fan status |
| hwstat | Hardware status |
| id | Device id string |
| lcdbl | LCD backlight |
| mode | Master/Slave mode |
| portctl | Communication port control flags |
| serial | Device serial number string |
| temp | Device internal temperature |
| version | Device firmware version |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## baud (baud rate)

This command may be used as a query to determine the baud rate setting for the serial port. It may also be used as an update to set the baud rate. The data is an integer type. The following values are allowed:

- 9600
- 19200
- 38400
- 57600

- 115200

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **baud?<CR>** | **OK 57600<CRLF>** |
| UPDATE | **baud=57600<CR>** | **OK<CRLF>** |

# desc (description)

This command may be used as a query to read the user defined device description. It may also be used as an update to set the description. The data is a string type, with a limit of 32 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `desc` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `desc` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `desc` command: `desc="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **desc?<CR>** | **OK "Aloha Room East"<CRLF>** |
| UPDATE | **desc="Courtroom #12"<CR>** | **OK<CRLF>** |

# fanstat (fan status)

This command may be used as a query to determine the status of the cooling fan. The data type is integer, and the value is a code representing one of the following states:

- **0** means that the fan is running, and temperature is within limits
- **1** means that the fan was not detected on powerup or failed its startup test

- **8** means that the fan is not running, but temperature is within limits

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **fanstat?<CR>** | **OK 0<CRLF>** |

# hwstat (hardware status)

This command may be used as a query to determine the status of device hardware. The data type is integer, and the value is a code representing one of the following states:

- **0** means normal operation
- **1** means that an over-temperature condition has been detected
- **2** means that the fan was not detected on powerup or failed its startup test
- **3** means that the real time clock (RTC) back-up battery is low

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **hwstat?<CR>** | **OK 0<CRLF>** |

# id (device id)

This command may be used as a query to read the device id string. This is the "name" of the device used by the ASPEN protocol and is always "SPN32i". The data is a string type.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **id?<CR>** | **OK "SPN32i"<CRLF>** |

# lcdbl (LCD backlight)

This command may be used as a query to read the LCD backlight on/off status, or as an update to set the status. The data type is integer, either "1" meaning that the LCD backlight is on, or "0" meaning that it is not.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **lcdbl?<CR>** | **OK 1<CRLF>** |
| UPDATE | **lcdbl=0<CR>** | **OK<CRLF>** |

# mode (Master/Slave mode)

This command may be used as a query to determine if the SPN32i is operating as a **Master** or as a **Slave**. The data type is integer, either "1" meaning Master mode, or "0" meaning Slave mode.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **mode?<CR>** | **OK 0<CRLF>** |

# portctl (communication port control flags)

This command may be used as a query to read the port control flags, or as an update to set them. The communication port is specified by using the address syntax. Addresses must be in the range 0 to 4, representing the one of the following:

- **0** - USB port
- **1** - RS232 port
- **2** - TCP port 1
- **3** - TCP port 2 (*firmware versions 1.4.0 and higher*)
- **4** - HTTP port (*firmware versions 1.4.0 and higher*)

The data type is integer, in the range 0 to 3. The value is a code representing the control settings for the communication port:

| Code | Port Setting |
|---|---|
| 0 | Port is disabled |
| 1 | Port is receive only |

| | |
|---|---|
| 2 | Port is send only |
| 3 | Port is send/receive |

If the port address is wildcarded, then the data type is an array of integer of size 4. By default, all communication ports are send/receive (full duplex) but in some advanced 3rd party control scenarios a port may need to be set otherwise. **Note: to preserve the ability to communicate with the device, changes to the USB port setting have no effect, the default of send/receive is always in force.**

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **portctl(1)?<CR>** | **OK 3<CRLF>** |
| QUERY | **portctl(*)?<CR>** | **OK {3,3,3,3}<CRLF>** |
| UPDATE | **portctl(1)=2<CR>** | **OK<CRLF>** |
| UPDATE | **portctl(*)={3,3,3,3}<CR>** | **OK<CRLF>** |

# serial (serial number)

This command may be used as a query to read the device serial number. The data is a string type.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **serial?<CR>** | **OK "5100101"<CRLF>** |

# temp (Device internal temperature)

This command may be used as a query to read the internal temperature of the SPN32i. The data type is integer, in degrees C.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **temp?<CR>** | **OK 32<CRLF>** |

# version (firmware version)

This command may be used as a query to read the device firmware version number. The data is a string type.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **version?<CR>** | **OK "1.0.1"<CRLF>** |

# SPN32i Audio Input Commands

| | |
|---|---|
| inact | Audio input activity status |
| incl | Audio input clipping status |
| indel | Audio input delay |
| ingn | Audio input gain |
| ingrp | Audio input group |
| ingrpadd | Audio input group add member |
| ingrplb | Audio input group label |
| ingrprem | Audio input group remove member |
| iniv | Audio input phase invert status |
| inlb | Audio input channel label |
| inlv | Audio input level (dBu RMS) |
| inlvraw | 'Raw' audio input level (dBu RMS) |
| inmt | Audio input mute status |
| inmttog | Audio input mute toggle |
| inph | Audio input phantom power status |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## inact (input activity)

This command may be used to determine if an audio input channel is active, meaning that it has less than 3 dB of attenuation assigned to it by the automixing algorithm for some particular mix bus channel. Two forms exist for this command:

- **Query for input channel activity on ANY mix bus** - The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that the channel is active on **at least one** of the mix buses or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 32.

- **Query for input channel activity on a SPECIFIC mix bus** - An mix bus channel/input channel pair is specified by using the 2 dimensional address syntax. Addresses for the first dimension (mix bus) must be in the range 1 to 48. Addresses for the second dimension (input channel) must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that the channel is active on **the specified** mix bus, or "0" meaning that it is not. If the input channel address is wildcarded, then the data type is an array of integer of size 32. The mix bus address may not be wildcarded.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **inact(5)?<CR>** | **OK 0<CRLF>** |
| QUERY | **inact(*)?<CR>** | **OK {0,1,0,...,0,0,0}<CRLF>** |
| QUERY | **inact(5,5)?<CR>** | **OK 0<CRLF>** |
| QUERY | **inact(5,*)?<CR>** | **OK {0,1,0,...,0,0,0}<CRLF>** |

# incl (input clipping)

This command may be used to determine if an audio input channel is clipping, meaning that it is being overdriven by the input signal to the point of saturating the analog to digital converter. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that the channel is in clipping, or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **incl(7)?<CR>** | **OK 1<CRLF>** |
| QUERY | **incl(*)?<CR>** | **OK {0,0,0,...,0,0,0}<CRLF>** |

# indel (input delay)

This command may be used as a query to read the input delay, or as an update to set the delay. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, in the range 0 to 200, representing the time delay in one half millisecond increments. If the channel address is wildcarded, then the data type is an array of integer of size 32. In this case the value **9999** may be used in an update to indicate that a particular input delay is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **indel(1)?<CR>** | **OK 42<CRLF>** |
| QUERY | **indel(*)?<CR>** | **OK {0,40,0,...,0,0,0}<CRLF>** |
| UPDATE | **indel(5)=30<CR>** | **OK<CRLF>** |
| UPDATE | **indel(*)={12,0,0,...,8,9999,9999}<CR>** | **OK<CRLF>** |

# ingn (input gain)

This command may be used as a query to read the input channel gain, or as an update to set the gain. The input channel or input group is specified by using the address syntax. Addresses for input channels must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for input groups must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher). The data type is integer, representing the gain in dB. For mic/line inputs 1 - 32 the range is -10 to +60 dB and for test signal inputs 33 - 36 the range is -70 to +20 dB. If the channel address is wildcarded, then the data type is an array of integer of size 36. In this case the value **99** may be used in an update to indicate that a particular rear panel input gain is to **remain unchanged** by the command. An input group is updated with a single gain value which is applied to all members of the group. The response to a input group query or *verbose* update is an array of integer of size 36, containing the gain values for *all* input channels, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **ingn(1)?<CR>** | **OK 42<CRLF>** |
| QUERY | **ingn(101)?<CR>** | **OK {40,40,50,...,0,0,0}<CRLF>** |
| QUERY | **ingn(*)?<CR>** | **OK {40,40,50,...,0,0,0}<CRLF>** |
| UPDATE | **ingn(5)=30<CR>** | **OK<CRLF>** |
| UPDATE | **ingn(101)=30<CR>** | **OK<CRLF>** |
| UPDATE | **ingn(*)={40,40,50,...,0,99,99}<CR>** | **OK<CRLF>** |

# ingrp (input group)

*This command is available in firmware versions 1.5.0 and higher.*

This command may be used as a query to read the input channel group definition, or as an update to set the group definition. The input group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a variable length. The values contained in the array represent the input channels which are *members* of the group, in the range 1 to 36 (test signal inputs 33 - 36 are included). The special array **{0}** indicates an *empty* group, one without members, and may be used in an update to clear a previously defined group. Once defined, the address of a group may be substituted for the address of an individual channel in gain control and mute commands. For example: `rpinmt(101)=1` mutes all input channels which are members of group 101. An error will result if the address of an *empty* group is used in a command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **ingrp(102)?<CR>** | **OK {1,2,3,5,7}<CRLF>** |
| QUERY | **ingrp(101)?<CR>** | **OK {0}<CRLF>**<br>**(group 101 is empty)** |
| UPDATE | **ingrp(105)={1,2,3}<CR>** | **OK<CRLF>** |
| UPDATE | **ingrp(105)={0}<CR>**<br>**(empty group as argument to clear group 105)** | **OK<CRLF>** |

# ingrpadd (input group add member)

*This command is available in firmware versions 1.5.9 and higher.*

This command may be used as an update to *add* one or more input channels to a group definition. The input group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a variable length. The values contained in the array represent the input channels which are added to the group, in the range 1 to 36 (test signal inputs 33 - 36 are included).

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **ingrpadd(105)={1,2,3}<CR>** | **OK<CRLF>** |

# ingrplb (input group label)

*This command is available in firmware versions 1.5.0 and higher.*

This command may be used as a query to read the input group text label, or as an update to set the label. The input group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is string, with a limit of 15 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `ingrplb` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `ingrplb` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `ingrplb` command: `ingrplb(1)="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized..

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **ingrplb(101)?<CR>** | **OK "Media Feeds"<CRLF>** |
| UPDATE | **ingrplb(102)="West Room"<CR>** | **OK<CRLF>** |

# ingrprem (input group remove member)

*This command is available in firmware versions 1.5.9 and higher.*

This command may be used as an update to *remove* one or more input channels from a group definition. The input group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a variable length. The values contained in the array represent the input channels which are removed from the group, in the range 1 to 36 (test signal inputs 33 - 36 are included).

Examples:

| | REQUEST | RESPONSE |
|---|---|---|

| | | |
|---|---|---|
| UPDATE | **ingrprem(105)={1}<CR>** | **OK<CRLF>** |

# iniv (input phase invert)

This command may be used as a query to read the input channel phase invert status, or as an update to set the status. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 36 (test signal inputs 33 - 36 are included). The data type is integer, either "1" meaning that the input has the audio phase inverted (shifted by 180 degrees), or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 36. In this case the value **99** may be used in an update to indicate that a particular input invert state is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **iniv(3)?<CR>** | **OK 0<CRLF>** |
| QUERY | **iniv(*)?<CR>** | **OK {0,1,0,...,0,0,0}<CRLF>** |
| UPDATE | **iniv(2)=1<CR>** | **OK<CRLF>** |
| UPDATE | **iniv(*)={0,0,1,...,99,99,99}<CR>** | **OK<CRLF>** |

# inlb (input channel label)

This command may be used as a query to read the input channel text label, or as an update to set the label. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is string, with a limit of 15 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `inlb` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `inlb` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `inlb` command: `inlb(1)="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized..

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **inlb(1)?<CR>** | **OK "Chairman"<CRLF>** |
| UPDATE | **inlb(2)="#3 West"<CR>** | **OK<CRLF>** |

# inlv (audio input level)

This command may be used as a query to read the input channel level. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 36 (test signal inputs 33 - 36 are included). The data type is integer, in the range -70 to +20, representing the RMS audio level in dBu as measured at the input to the auto mixing matrix, after DSP processing is applied. If the channel address is wildcarded, then the data type is an array of integer of size 36.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **inlv(1)?<CR>** | **OK -23<CRLF>** |
| QUERY | **inlv(*)?<CR>** | **OK {-2,4,-10,...,-53,-71,-60}<CRLF>** |

# inlvraw ('raw' audio input level)

This command may be used as a query to read the input channel 'raw' level. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 36 (test signal inputs 33 - 36 are included). The data type is integer, in the range -70 to +20, representing the RMS audio level in dBu as measured after the input preamplifier but before DSP is applied. If the channel address is wildcarded, then the data type is an array of integer of size 36.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **inlvraw(1)?<CR>** | **OK -23<CRLF>** |
| QUERY | **inlvraw(*)?<CR>** | **OK {-2,4,-10,...,-53,-71,-60}<CRLF>** |

# inmt (input mute)

This command may be used as a query to read the input channel mute status, or as an update to set the status. The input channel or input group is specified by using the address syntax. Addresses for input channels must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for input groups must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher). The data type is integer, either "1" meaning that the input is muted, or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 36. In this case the value **99** may be used in an update to indicate that a particular input mute is to **remain unchanged** by the command. An input group is updated with a single mute value which is applied to all members of the group. The response to a input group query or *verbose* update is an array of integer of size 36, containing the mute values for *all* input channels, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **inmt(3)?<CR>** | **OK 1<CRLF>** |
| QUERY | **inmt(101)?<CR>** | **OK {0,0,0,...,0,1,0}<CRLF>** |
| QUERY | **inmt(*)?<CR>** | **OK {0,0,0,...,0,1,0}<CRLF>** |
| UPDATE | **inmt(2)=0<CR>** | **OK<CRLF>** |
| UPDATE | **inmt(101)=0<CR>** | **OK<CRLF>** |
| UPDATE | **inmt(*)={0,0,0,...,99,99,0}<CR>** | **OK<CRLF>** |

# inmttog (input mute toggle)

This command may be used as a simple comand to toggle input channel mute status. The input channel or input group is specified by using the address syntax. Addresses for input channels must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for input groups must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher). The response to a input group *verbose* command is an array of integer of size 36, containing the mute values for *all* input channels, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **inmttog(4)<CR>** | **OK<CRLF>** |
| COMMAND | **inmttog(101)<CR>** | **OK<CRLF>** |

# inph (input phantom power)

This command may be used as a query to read the input phantom power status, or as an update to set the status. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that phantom power is enabled, or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **inph(3)?<CR>** | **OK 1<CRLF>** |
| QUERY | **inph(*)?<CR>** | **OK {1,1,1,...,0,0,0}<CRLF>** |
| UPDATE | **inph(4)=0<CR>** | **OK<CRLF>** |
| UPDATE | **inph(*)={0,1,1,...,1,1,1}<CR>** | **OK<CRLF>** |

# SPN32i Input Compressor Commands

| | |
|---|---|
| incpgn | Input compressor gain |
| incpmug | Input compressor makeup gain |
| incprat | Input compressor ratio |
| incptc | Input compressor time constant |
| incpthr | Input compressor threshold level |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## incpgn (input compressor gain)

This command may be used as a query to read the compressor gain. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, representing the gain in dB, which is always a negative value if the compressor is active, or zero. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`incpgn(1)?<CR>`** | **`OK -5<CRLF>`** |
| QUERY | **`incpgn(*)?<CR>`** | **`OK {-5,0,0,...,0,0,0}<CRLF>`** |

## incpmug (input compressor makeup gain)

This command may be used as a query to read the makeup gain, or as an update to set the makeup gain. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, in the range 0 to 30, representing the gain in dB. If the channel address is wildcarded, then the data type is an array of integer of size 32. The allowable upper limit of makeup gain values is dynamically determined by the threshold and ratio settings of the compressor, and may be less than the absolute maximum of 30 dB. If the value sent in an update request exceeds this upper limit, it will be forced to the limit value.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **incpmug(1)?<CR>** | **OK 5<CRLF>** |
| QUERY | **incpmug(*)?<CR>** | **OK {3,3,3,...,0,0,0}<CRLF>** |
| UPDATE | **incpmug(5)=10<CR>** | **OK<CRLF>** |
| UPDATE | **incpmug(*)={6,6,6,...,0,0,0}<CR>** | **OK<CRLF>** |

# incprat (input compressor ratio)

This command may be used as a query to read the ratio, or as an update to set the ratio. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is float, in the range 0.0 to 50.0, representing the compression ratio, with 0.0 meaning that the compressor is "off". If the channel address is wildcarded, then the data type is an array of float of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **incprat(1)?<CR>** | **OK 4.5<CRLF>** |
| QUERY | **incprat(*)?<CR>** | **OK {3.0,3.0,...,0.0,0.0}<CRLF>** |
| UPDATE | **incprat(5)=5.0<CR>** | **OK<CRLF>** |
| UPDATE | **incprat(*)={3.0,3.0,...,0.0,0.0}<CR>** | **OK<CRLF>** |

# incptc (input compressor time constant)

This command may be used as a query to read the time constant, or as an update to set the time constant. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, in the range 5 to

10000, representing the time in one tenth millisecond increments. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **incptc(1)?<CR>** | **OK 1000<CRLF>** |
| QUERY | **incptc(*)?<CR>** | **OK {2000,2000,...,2000,2000}<CRLF>** |
| UPDATE | **incptc(3)=2000<CR>** | **OK<CRLF>** |
| UPDATE | **incptc(*)={2000,2000,...,2000,2000}<CR>** | **OK<CRLF>** |

# incpthr (input compressor threshold level)

This command may be used as a query to read the threshold, or as an update to set the threshold. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, in the range -80 to +20, representing the level in dBu. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **incpthr(1)?<CR>** | **OK -3<CRLF>** |
| QUERY | **incpthr(*)?<CR>** | **OK {-3,0,0,...,0,0,0}<CRLF>** |
| UPDATE | **incpthr(5)=-20<CR>** | **OK<CRLF>** |
| UPDATE | **incpthr(*)={-3,0,0,...,0,0,0}<CR>** | **OK<CRLF>** |

# SPN32i Input EQ Filter Commands

| | |
|---|---|
| infil | Input EQ filter definition |
| infilby | Input EQ filter bypass status |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## infil (input EQ filter parameters)

This command may be used as a query to read an input EQ filter definition, or as an update to modify the definition. The filter is specified by using the 2 dimensional address syntax. Addresses for the first dimension (input channel) must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). Addresses for the second dimension (EQ filter stage) must be in the range 1 to 4. Wildcards may **not** be used for either dimension. The data type is array of integer, with a fixed length of 6. The values contained in the array represent the 6 parameters of the EQ filter, using the following scheme:

- The **first** integer is a code that specifies the **type** of the filter. It may be in the range 0 to 5, with the following meanings:

| Code | Filter Type |
|---|---|
| 0 | All Pass (no filtering) |
| 1 | Low Pass |
| 2 | High Pass |
| 3 | Low Shelving |
| 4 | High Shelving |
| 5 | Parametric |

- The **second** integer is a code that specifies the **slope** of the filter. It may be in the range 1 to 4, with the following meanings:

| Code | Filter Slope |
|---|---|
| 1 | 6dB per octave (1st order) |
| 2 | 12dB per octave (2nd order) |
| 3 | 18dB per octave (3rd order) |
| 4 | 24dB per octave (4th order) |

    - **RULE:** The filter order parameter applies to the low/high shelving and low/high pass filter types only. Use 0 (zero) as a placeholder for the remaining filter types.
- The **third** integer is a code that specifies the **approximation** of the filter. It may be in the range 0 to 2, with the following meanings:

| Code | Filter Approximation |
|---|---|
| 0 | Butterworth |
| 1 | Bessel |
| 2 | Linkwitz-Riley |

    - **RULE:** The filter approximation parameter applies to the low/high shelving and low/high pass filter types **only**. Use 0 (zero) as a placeholder for the remaining filter types.
    - **RULE:** The Linkwitz-Riley filter approximation may be applied to the low/high pass filter types **only**, with the further limitation that these filters have slopes of 12dB or 24dB per octave **only** (2nd or 4th order filters only)**.**
- The **fourth** integer specifies the **corner frequency** of the filter in Hz. It may be in the range 20 to 20000.
    - **RULE:** The filter frequency parameter applies to all filters **except** the all pass filter. Use 0 (zero) as a placeholder for the all pass filter type.
- The **fifth** integer specifies the **gain** (boost/cut) of the filter in dB. It may be in the range -30 to 30.
    - **RULE:** The filter gain parameter applies to the low/high shelving and parametric filter types **only**. Use 0 (zero) as a placeholder for the remaining filter types.
- The **sixth** integer specifies the **bandwidth** of the filter in hundredths of an octave (octaves / 100). It may be in the range 5 to 1000.
    - **RULE:** The filter bandwidth parameter applies to the parametric filter type **only**. Use 0 (zero) as a placeholder for the remaining filter types.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **infil(5,1)?<CR>** | **OK {5,1,0,1000,3,100}<CRLF>** |
| UPDATE | **infil(6,1)={5,1,0,1000,3,100}<CR>** | **OK<CRLF>** |

# infilby (input EQ filter bypass status)

This command may be used as a query to read the EQ filter bypass status, or as an update to set the status. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that EQ filtering on the input is bypassed (all stages bypassed), or "0" meaning that it is not bypassed (all stages applied). If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **infilby(1)?<CR>** | **OK 0<CRLF>** |
| QUERY | **infilby(*)?<CR>** | **OK {0,0,0,...,0,0,0}<CRLF>** |
| UPDATE | **infilby(5)=1<CR>** | **OK<CRLF>** |
| UPDATE | **infilby(*)={1,1,1,...,0,0,0}<CR>** | **OK<CRLF>** |

# SPN32i ADFE (Automatic Digital Feedback Elimination) Commands

| | |
|---|---|
| adfeen | ADFE enable status |
| adfefil | ADFE filter deployment |
| adfelten | ADFE auto-lift enable status |
| adfelttm | ADFE auto-lift time |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## adfeen (adfe enable status)

This command may be used as a query to read the adfe enable status, or as an update to set the status. The audio input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that the adfe feature is enabled, or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`adfeen(2)?<CR>`** | **`OK 1<CRLF>`** |
| QUERY | **`adfeen(*)?<CR>`** | **`OK OK {1,1,1,...,1,1,1}<CRLF>`** |
| UPDATE | **`adfeen(7)=0<CR>`** | **`OK<CRLF>`** |
| UPDATE | **`adfeen(*)=`**`{1,1,1,...,1,1,1}<CR>` | **`OK<CRLF>`** |

# adfefil (adfe filter deployment)

This command may be used as a query to read the adfe filter deployment status, or as an update to deploy an adfe filter. The input channel and filter index are specified by using the 2 dimensional address syntax. Addresses for the first dimension (input channel) must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). Addresses for the second dimension (adfe filter index) must be in the range 1 to 8. The data type is integer, unless the adfe filter index is wildcarded, in which case the data type is an array of integer of size 8. This is allowed for queries only. The data values are in the range 20 to 20000, where 0 means that no filter is deployed, and the others represent the center frequency of the adfe filter in Hz.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **adfefil(3,2)?<CR>** | **OK 0<CRLF>** |
| QUERY | **adfefil(5,*)?<CR>** | **OK {12,0,0,0,0,0,0,0}<CRLF>** |
| UPDATE | **adfefil(4,1)=15<CR>** | **OK<CRLF>** |

# adfelten (adfe auto-lift enable status)

This command may be used as a query to read the adfe auto-lift enable status, or as an update to set the status. The audio input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that the adfe auto-lift feature is enabled, or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **adfelten(2)?<CR>** | **OK 1<CRLF>** |
| QUERY | **adfelten(*)?<CR>** | **OK {1,1,1,...,1,1,1}<CRLF>** |
| UPDATE | **adfelten(7)=0<CR>** | **OK<CRLF>** |
| UPDATE | **adfelten(*)={1,1,1,...,1,1,1}<CR>** | **OK<CRLF>** |

# adfelttm (adfe auto-lift time)

This command may be used as a query to read the adfe auto-lift time, or as an update to set the auto-lift time. The audio input channel is specified by using the address syntax. Addresses must

be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, in the range 10 to 120, representing the auto-lift time in minutes. If the channel address is wildcarded, then the data type is an array of integer of size 32.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **adfelttm(2)?<CR>** | **OK 10<CRLF>** |
| QUERY | **adfelttm(*)?<CR>** | **OK {30,30,30,...,30,30,30}<CRLF>** |
| UPDATE | **adfelttm(7)=0<CR>** | **OK<CRLF>** |
| UPDATE | **adfelttm(*)={30,30,30,...,30,30,30}<CR>** | **OK<CRLF>** |

# SPN32i Audio
# Input Noise Reduction Filter (NRF) Commands

| | |
|---|---|
| nrfdep | Audio input NRF depth |
| nrfen | Audio input NRF enable status |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## nrfdep (input NRF depth)

This command may be used as a query to read the input channel NRF depth, or as an update to set the depth. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, in the range 6 to 36, representing the depth (attenuation) of the noise reduction filter in dB. If the channel address is wildcarded, then the data type is an array of integer of size 32. In this case the value **99** may be used in an update to indicate that a particular input gain is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`nrfdep(1)?<CR>`** | **`OK 24<CRLF>`** |
| QUERY | **`nrfdep(*)?<CR>`** | **`OK {24,24,24,...,24,24,24}<CRLF>`** |
| UPDATE | **`nrfdep(5)=20<CR>`** | **`OK<CRLF>`** |
| UPDATE | **`nrfdep(*)={30,30,30,...,99,99,99}<CR>`** | **`OK<CRLF>`** |

# nrfen (input NRF enable status)

This command may be used as a query to read the input channel NRF enable status, or as an update to set the status. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The data type is integer, either "1" meaning that noise reduction filtering on the input is enabled, or "0" meaning that it is disabled. If the channel address is wildcarded, then the data type is an array of integer of size 32. In this case the value **99** may be used in an update to indicate that a particular input invert state is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **nrfen(3)?<CR>** | **OK 0<CRLF>** |
| QUERY | **nrfen(*)?<CR>** | **OK {0,1,0,...,0,0,0}<CRLF>** |
| UPDATE | **nrfen(2)=1<CR>** | **OK<CRLF>** |
| UPDATE | **nrfen(*)={0,0,1,...,99,99,99}<CR>** | **OK<CRLF>** |

# SPN32i Matrix Crosspoint Commands

| | |
|---|---|
| xpgn | Matrix crosspoint gain |
| xpgnst | Matrix crosspoint gain step |
| xpgrp | Matrix crosspoint group |
| xpgrpadd | Matrix crosspoint group add member |
| xpgrplb | Matrix crosspoint group label |
| xpgrprem | Matrix crosspoint group remove member |
| xpiv | Matrix crosspoint phase invert status |
| xpmode | Matrix crosspoint mix mode |
| xpmt | Matrix crosspoint mute status |
| xpmttog | Matrix crosspoint mute toggle |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## xpgn (crosspoint gain)

This command may be used as a query to read the matrix crosspoint gain, or as an update to set the gain. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address.

In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire *column* of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire *row* of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon

character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`. Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded.

In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher).

The data type is integer, in the range -70 to +20, representing the gain in dB. The value -70 has the special meaning **Off**. If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. In these cases the value **99** may be used in an update to indicate that a particular crosspoint gain is to **remain unchanged** by the command. A crosspoint group is updated with a single gain value which is applied to all members of the group. The response to a crosspoint group query or *verbose* update is an array of size 48 if the group has been defined as a *Row* type, or an array of size 36 if it has been defined as a *Column* type. The array contains the gain values for *all* crosspoints in the row or column, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **xpgn(1,7)?<CR>** | **OK -3<CRLF>** |
| QUERY | **xpgn(101)?<CR>** | **OK {0,3,0,...0,0,0}<CRLF>** |
| QUERY | **xpgn(*,1)?<CR>** | **OK {0,3,0,...0,0,0}<CRLF>** |
| QUERY | **xpgn(1,*)?<CR>** | **OK {0,3,0,...0,0,0}<CRLF>** |
| QUERY | **xpgn(1:3,7)?<CR>** | **OK {0,3,0}<CRLF>** |
| QUERY | **xpgn(1,10:15)?<CR>** | **OK {0,3,0,0,0,0}<CRLF>** |
| UPDATE | **xpgn(5,2)=5<CR>** | **OK<CRLF>** |
| UPDATE | **xpgn(101)=5<CR>** | **OK<CRLF>** |
| UPDATE | **xpgn(*,5)={0,5,3,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **xpgn(5,*)={0,5,3,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **xpgn(1:3,7)={0,5,3}<CR>** | **OK<CRLF>** |
| UPDATE | **xpgn(1,10:15)={0,5,3,0,0,0}<CR>** | **OK<CRLF>** |

# xpgnst ( crosspoint gain step change)

This command is used as an update to step the gain by the amount specified, either up or down. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address.

In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire *column* of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire *row* of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`. Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded.

In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher).

The data type is integer, in the range -6 to +6, representing the gain step in dB. A positive value increments the gain, a negative value decrements the gain. If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. A crosspoint group is updated with a single gain step value which is applied to all members of the group. The response to a crosspoint group *verbose* update is an array of size 48 if the group has been defined as a *Row* type, or an array of size 36 if it has been defined as a *Column* type. The array contains the gain values for *all* crosspoints in the row or column, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **xpgnst(2,2)=2<CR>** | **OK<CRLF>** |
| UPDATE | **xpgnst(101)=2<CR>** | **OK<CRLF>** |
| UPDATE | **xpgnst(*,3)={2,2,2,...0,0,0}<CR>** | **OK<CRLF>** |
| UPDATE | **xpgnst(3,*)={-1,-1,-1,...0,0,0}<CR>** | **OK<CRLF>** |
| UPDATE | **xpgnst(3:8,12)={-1,-1,-1,-1,-1,-1}<CR>** | **OK<CRLF>** |
| UPDATE | **xpgnst(3,5:8)={-1,-1,-1,-1}<CR>** | **OK<CRLF>** |

# xpgrp (input group)

*This command is available in firmware versions 1.5.0 and higher.*

This command may be used as a query to read the crosspoint group definition, or as an update to set the group definition. The crosspoint group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a variable length.

- The **first** integer is a code that specifies the **type** of crosspoint group being defined. It may be in the range 0 to 2, with the following meanings:

| Code | Crosspoint Group Type |
|---|---|
| 0 | Empty (no members) |
| 1 | Row (crosspoints associated with a single input channel) |
| 2 | Column (crosspoints associated with a single mix bus) |

- The **second** integer specifies a *input channel* if the group type is **Row**, or a *mix bus* if the group type is **Column**. It may be in the range 1 to 36 for input channels, and in the range 1 to 48 for mix busses. *Not present if the type is **Empty**.*
- The **remaining** values specify *mix bus* addresses if the group type is **Row**, or *input channel* addresses if the group type is **Column**. They may be in the range 1 to 36 for input channels, and in the range 1 to 48 for mix busses. *Not present if the type is **Empty**.*

The values contained in the array define which crosspoints are *members* of the group. The special array **{0}** indicates an *empty* group, one without members, and may be used in an update to clear a previously defined group. Once defined, the address of a group may be substituted for the address of an individual channel in gain control and mute commands. For example: `rpxpmt(101)=1` mutes all crosspoints which are members of group 101. An error will result if the address of an *empty* group is used in a command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **xpgrp(102)?<CR>** | **OK {1,5,1,2}<CRLF> (type is Row, input is 5, group members are crosspoints 5,1 and 5,2)** |
| QUERY | **xpgrp(101)?<CR>** | **OK {0}<CRLF> (group 101 is empty)** |
| UPDATE | **xpgrp(105)={2,17,3,4}<CR> (type is Column, mix bus is 17, group members are crosspoints 3,17 and 4,17)** | **OK<CRLF>** |
| UPDATE | **xpgrp(105)={0}<CR> (empty group as argument to clear group 105)** | **OK<CRLF>** |

# xpgrpadd (crosspoint group add member)

*This command is available in firmware versions 1.5.9 and higher.*

This command may be used as an update to *add* one or more crosspoints to an existing (non-empty) group definition. The crosspoint group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a variable length. The values contained in the array represent crosspoints which are added to the group, interpreted in accordance with the crosspoint group type. For the **Row** type values must be in the range 1 to 36 (input channels). For the **Column** type values must be in the range 1 to 48 (mix busses). An error will result if an *empty* group is addressed by this command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **xpgrpadd(105)={10}<CR>** | **OK<CRLF>** |

# xpgrplb (crosspoint group label)

*This command is available in firmware versions 1.5.0 and higher.*

This command may be used as a query to read the crosspoint group text label, or as an update to set the label. The crosspoint group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is string, with a limit of 15 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `xpgrplb` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `xpgrplb` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `xpgrplb` command: `xpgrplb(1)="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized..

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **xpgrplb(101)?<CR>** | **OK "Media Feeds"<CRLF>** |
| UPDATE | **xpgrplb(102)="West Room"<CR>** | **OK<CRLF>** |

# xpgrprem (crosspoint group remove member)

*This command is available in firmware versions 1.5.9 and higher.*

This command may be used as an update to *remove* one or more crosspoints from an existing (non-empty) group definition. The crosspoint group is specified by using the address syntax. Addresses must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a variable length. The values contained in the array represent crosspoints which are removed from the group, interpreted in accordance with the crosspoint group type. For the **Row** type values must be in the range 1 to 36 (input channels). For the **Column** type values must be in the range 1 to 48 (mix busses). An error will result if an *empty* group is addressed by this command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **xpgrprem(105)={10}<CR>** | **OK<CRLF>** |

# xpiv (crosspoint phase invert status)

This command may be used as a query to read the matrix crosspoint phase invert status, or as an update to set the status. The crosspoint is specified by using the 2 dimensional address syntax. Addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included), or wildcarded. Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire column of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire row of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`. Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded. The data type is integer, either "1" meaning that the crosspoint is muted, or "0" meaning that it is not. If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. In these cases the value **99** may be used in an update to indicate that a particular crosspoint mute is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **xpiv(3,9)?<CR>** | **OK 1<CRLF>** |
| QUERY | **xpiv(*,5)?<CR>** | **OK {0,0,0,...1,1,0}<CRLF>** |

| | | |
|---|---|---|
| QUERY | **xpiv(5,*)?<CR>** | **OK {0,0,0,...1,1,0}<CRLF>** |
| QUERY | **xpiv(5,10:15)?<CR>** | **OK {0,0,0,1,1,0}<CRLF>** |
| QUERY | **xpiv(3:4,10)?<CR>** | **OK {0,0}<CRLF>** |
| UPDATE | **xpiv(1,3)=0<CR>** | **OK<CRLF>** |
| UPDATE | **xpiv(*,5)={0,1,0,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **xpiv(5,*)={0,1,0,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **xpiv(1:5,3)={0,0,0,0,0}<CR>** | **OK<CRLF>** |
| UPDATE | **xpiv(1,5:7)={1,1,1}<CR>** | **OK<CRLF>** |

# xpmode (crosspoint mix mode)

This command may be used as a query to read the matrix crosspoint mix mode, or as an update to set the mode. The crosspoint is specified by using the 2 dimensional address syntax. Addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included), or wildcarded. Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire column of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire row of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`. Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded. The data type is integer, with the following values allowed:

- **0** means that the crosspoint is in DIRECT mode
- **1** means that the crosspoint is in OVERRIDE mode
- **2** means that the crosspoint is in BACKGROUND mode
- **3** means that the crosspoint is in AUTO mode
- **5** means that the crosspoint is in PHANTOM mode

If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. In these cases the value **99** may be used in an update to indicate that a particular crosspoint mix mode is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|

| | | |
|---|---|---|
| QUERY | **xpmode(5,9)?<CR>** | **OK 3<CRLF>** |
| QUERY | **xpmode(*,1)?<CR>** | **OK {3,3,0,...3,3,3}<CRLF>** |
| QUERY | **xpmode(1,*)?<CR>** | **OK {3,3,0,...3,3,3}<CRLF>** |
| QUERY | **xpmode(1:3,1)?<CR>** | **OK {3,3,0}<CRLF>** |
| QUERY | **xpmode(5,12:16)?<CR>** | **OK {3,3,0,3,3}<CRLF>** |
| UPDATE | **xpmode(8,1)=0<CR>** | **OK<CRLF>** |
| UPDATE | **xpmode(*,5)={3,1,3,...99,99,3}<CR>** | **OK<CRLF>** |
| UPDATE | **xpmode(5,*)={3,1,3,...99,99,3}<CR>** | **OK<CRLF>** |
| UPDATE | **xpmode(1:3,1)={3,1,3}<CR>** | **OK<CRLF>** |
| UPDATE | **xpmode(5,12:16)={3,1,3,1,3}<CR>** | **OK<CRLF>** |

# xpmt (crosspoint mute)

This command may be used as a query to read the matrix crosspoint mute status, or as an update to set the status. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address.

In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire *column* of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire *row* of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`. Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded.

In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher).

The data type is integer, either "1" meaning that the crosspoint is muted, or "0" meaning that it is not. If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. In these cases the value **99** may be used in an update to indicate that a particular crosspoint mute is to **remain unchanged** by the command. A crosspoint group is updated with a single mute value which is applied to all members of the group. The response to a crosspoint group query or *verbose* update is an array of size 48 if the group has been defined as a *Row* type, or an array of size 36 if it has been defined as a *Column* type. The array contains the mute values for *all* crosspoints in the row or column, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **xpmt(3,9)?<CR>** | **OK 1<CRLF>** |
| QUERY | **xpmt(101)?<CR>** | **OK {0,0,0,...1,1,0}<CRLF>** |
| QUERY | **xpmt(*,5)?<CR>** | **OK {0,0,0,...1,1,0}<CRLF>** |
| QUERY | **xpmt(5,*)?<CR>** | **OK {0,0,0,...1,1,0}<CRLF>** |
| QUERY | **xpmt(5,10:15)?<CR>** | **OK {0,0,0,1,1,0}<CRLF>** |
| QUERY | **xpmt(3:4,10)?<CR>** | **OK {0,0}<CRLF>** |
| UPDATE | **xpmt(1,3)=0<CR>** | **OK<CRLF>** |
| UPDATE | **xpmt(101)=0<CR>** | **OK<CRLF>** |
| UPDATE | **xpmt(*,5)={0,1,0,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **xpmt(5,*)={0,1,0,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **xpmt(1:5,3)={0,0,0,0,0}<CR>** | **OK<CRLF>** |
| UPDATE | **xpmt(1,5:7)={1,1,1}<CR>** | **OK<CRLF>** |

# xpmttog (crosspoint mute toggle)

This command may be used as a simple command to toggle the matrix crosspoint mute status. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address.

In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48. In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher). The response to a crosspoint group *verbose* command is an array of size 48 if the group has been defined as a *Row* type, or an array of size 36 if it has been defined as a *Column* type. The array contains the gain values for *all* crosspoints in the row or column, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **xpmttog(3,9)<CR>** | **OK<CRLF>** |
| COMMAND | **xpmttog(101)<CR>** | **OK<CRLF>** |

# SPN32i Mix Bus Commands

| | |
|---|---|
| mixlb | Mix Bus label |
| mixlv | Final Mix level (dBu RMS) |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## mixlb (mix bus label)

This command may be used as a query to read the mix bus text label, or as an update to set the label. The mix bus is specified by using the address syntax. Addresses must be in the range 1 to 48. The data type is string, with a limit of 15 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `outlb` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `outlb` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `outlb` command: `outlb(1)="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|

| | | |
|---|---|---|
| QUERY | **mixlb(1)?<CR>** | **OK "Zone 3"<CRLF>** |
| UPDATE | **mixlb(2)="Gallery"<CR>** | **OK<CRLF>** |

# mixlv (final mix level)

This command may be used as a query to read the final mix level. The final mix bus is specified by using the address syntax. Addresses must be in the range 1 to 48. The data type is integer, in the range -70 to +20, representing the RMS audio level in dBu. If the channel address is wildcarded, then the data type is an array of integer of size 48.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **mixlv(1)?<CR>** | **OK -23<CRLF>** |
| QUERY | **mixlv(*)?<CR>** | **OK {3,-4,-66,...,-80,-80,4}<CRLF>** |

# SPN32i Programmable I/O Commands

| | |
|---|---|
| prgin | Programmable input state |
| prgindef | Programmable input definition |
| prginundef | Programmable input un-definition |
| prgout | Programmable output state |
| prgoutdef | Programmable output definition |
| prgoutht | Programmable output hold time |
| prgoutiv | Programmable output invert |
| prgoutpl | Programmable output pulse |
| prgoutqt | Programmable output qualification time |
| vprgin | Virtual programmable input state |
| vprgindef | Virtual programmable input definition |
| vprginundef | Virtual programmable input un-definition |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## prgin (programmable input state)

This command may be used to simulate a **momentary** contact closure on a programmable input or as a query to read the state of programmable inputs. The programmable input is specified by using the address syntax. Addresses must be in the range 1 to 30. **Note:** simulated contact closures have no effect for programmable inputs configured for **analog gain control**. The data type returned by queries is integer, with the possible values depending on the nature of the function assigned to the programmable input. For **analog gain control** functions the values are the range 0 to 255, representing the voltage sensed by the programmable input (0 is 0 volts, 255 is 5 volts). For all other functions the value is either "1", meaning that the input is active

(closed), or "0" meaning that it is not (open). The channel address may be wildcarded in queries, in which case the data type is an array of integer of size 30.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **prgin(11)<CR>** | **OK<CRLF>** |
| QUERY | **prgin(9)?<CR>** | **OK 1<CRLF>** |
| QUERY | **prgin(*)?<CR>** | **OK {0,0,1,...,0,0,0}<CRLF>** |

# prgindef (programmable input definition)

This command may be used as a query to read a programmable input definition, or as an update to modify the definition. The programmable input is specified by using the address syntax. Addresses must be in the range 1 to 30. The data type is array of integer, with a variable length. The values contained in the array represent the definition of a programmable input, using the following scheme:

- The **first** integer is a code that specifies the **function** that is assigned to the programmable input. It may be in the range 0 to 19, with the following meanings:

| Code | Function | Code | Function |
|---|---|---|---|
| 0 | No function assigned | 13 | Momentary mute on crosspoint |
| 1 | Analog input gain control | 14 | Momentary unmute on input (PTT) |
| 2 | Increment input gain 1 dB | 15 | Analog crosspoint gain control |
| 3 | Decrement input gain 1 dB | 16 | Increment crosspoint gain 1 dB |
| 7 | Recall preset from memory | 17 | Decrement crosspoint gain 1 dB |
| 8 | Toggle mute on input | 18 | Run macros on close |
| 10 | Toggle mute on crosspoint | 19 | Run macros on close/open |
| 11 | Momentary mute on input | | |

- The choice of function determines what happens when the programmable input is **asserted**. The choice of function affects the interpretation of the subsequent values in the array, which identify the **target** of the function.
  **NOTE:** codes 4, 5, 6 and 9 are not used by the SPN32i.
  **NOTE:** Functions which control matrix crosspoints are limited to affecting a group of crosspoints that are members of a single **row** of the matrix. A row consists of all crosspoints associated with one particular **input** channel.
  **NOTE:** All gain changes made by programmable input functions affect **rear panel** gain only.

- The **remaining** values specify the **target** of the function. These values are simply a list of one or more integers which identifies the thing(s) that are controlled by the function. The interpretation of the target values depends on the function that is assigned:

| Function | Target values |
|---|---|
| Analog input gain control | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Increment input gain 1 dB | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Decrement input gain 1 dB | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Recall preset from memory | A single integer value in the range 1 to 24. This identifies which preset is to be recalled by the function. |
| Toggle mute on input | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Toggle mute on crosspoint | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Momentary mute on input | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Momentary mute on crosspoint | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Momentary unmute on input (PTT) | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Analog crosspoint gain control | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Increment crosspoint gain 1 dB | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Decrement crosspoint gain 1 dB | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Run macro on close | A single integer value in the range 1 to 256. This identifies which macro is to be run by the function when the programmable input is asserted. |

| | |
|---|---|
| Run macros on close/open | Two integer values in the range 1 to 256. The first identifies which macro is to be run by the function when the programmable input is asserted. The second identifies which macro is to be run when the programmable input is subsequently de-asserted. |

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **prgindef(9)?<CR>** | **OK {1,1,3,4,10}<CRLF>** (Function "Analog input gain control" is assigned to programmable input 9. The input channels under control are 1, 3, 4, and 10.) |
| QUERY | **prgindef(2)?<CR>** | **OK {14,22}<CRLF>** (Function "Run a macro" is assigned to programmable input 2. Macro number 22 will be run if the programmable input is asserted.) |
| QUERY | **prgindef(3)?<CR>** | **OK {13,3,1,7,8}<CRLF>** (Function "Momentary mute on crosspoint" is assigned to programmable input 3. The crosspoints under control are {3,1}, {3,7}, and {3,8}, which are members of the matrix row associated with input channel 3.) |
| QUERY | **prgindef(4)?<CR>** | **OK {0}<CRLF>** (No function is assigned to programmable input 4.) |
| UPDATE | **prgindef(11)={7,5}<CR>** (Function "Recall preset from memory" is assigned to programmable input 11. The preset to be recalled is specified as number 5.) | **OK<CRLF>** |
| UPDATE | **prgindef(2)={8,1,2,4,5,6}<CR>** (Function "Toggle mute on input" is assigned to programmable input 2. The inputs to be controlled are specified as 1, 2, 4, 5, and 6.) | **OK<CRLF>** |
| UPDATE | **prgindef(4)={0}<CR>** ("No function" is assigned to programmable input 4, so that asserting it will have no effect.) | **OK<CRLF>** |
| UPDATE | **prgindef(1)={10,5,9,10}<CR>** (Function "Toggle mute on crosspoint" is assigned to programmable input 1. The crosspoints to be controlled are specified as | **OK<CRLF>** |

| | | |
|---|---|---|
| | {5,9} and {5,10}, which are members of the matrix row associated with input channel 5.) | |

# prginundef (programmable input un-definition)

This command may be used as an update to **un-define** one or more programmable inputs, meaning that it no longer has any function assigned to it. The data type is array of integer, with a variable length. The values contained in the array represent the address of a programmable input to be un-defined, in the range 1 to 30.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **prginundef={3,4,5}<CR>**<br>(Programmable inputs 3, 4, and 5 will be un-defined.) | **OK<CRLF>** |

# prgout (programmable output state)

This command may be used as a query to read a programmable output state, or as an update to set the state. The programmable output is specified by using the address syntax. Addresses must be in the range 1 to 16. The data type is integer, with the value either "1", meaning that the output is active, or "0" meaning that it is not. If the channel address is wildcarded, then the data type is an array of integer of size 16. In this case the value **99** may be used to indicate that a particular programmable output state is to **remain unchanged** by the command. **Note:** a programmable output may be set with an update **only** if no function is assigned to it. If a function is assigned, the update attempt will be ignored.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **prgout(5)?<CR>** | **OK 0<CRLF>** |
| QUERY | **prgout(*)?<CR>** | **OK {0,0,1,...,0,0,0}<CRLF>** |
| UPDATE | **prgout(2)=1<CR>** | **OK<CRLF>** |
| UPDATE | **prgout(*)={0,1,1,...,99,99,99}<CR>** | **OK<CRLF>** |

# prgoutdef (programmable output definition)

This command may be used as a query to read a programmable output definition, or as an update to modify the definition. The programmable output is specified by using the address syntax. Addresses must be in the range 1 to 16. The data type is array of integer, with a variable length. The values contained in the array represent the definition of a programmable output, using the following scheme:

- The **first** integer is a code that specifies the **function** that is assigned to the programmable output. It may be in the range 0 to 5, with the following meanings:

| Code | Function | Code | Function |
|---|---|---|---|
| 0 | No function assigned | 3 | Monitor active preset |
| 1 | Monitor audio input activity | 4 | Monitor hardware status |
| 2 | Monitor programmable input state | 5 | Monitor virtual programmable input state (*firmware versions 1.4.0 and higher*) |

- The choice of function determines what conditions or events control the state of the programmable output. The choice of function affects the interpretation of the subsequent values in the array, which identify the event **source** which is monitored by the function.

- The **remaining** values specify the **source** of the function. These values are simply a list of one or more integers which identifies the thing(s) that are controlled by the function. The interpretation of the source values depends on the function that is assigned:

| Function | Target values |
|---|---|
| Monitor audio input activity | A single integer in the range 1 to 49, followed by a sequence of zero or more integers in the range 1 to 32 (test signal inputs 33 - 36 are excluded). The first identifies which mix bus (NOM bus) channel is to be referenced by the function. The special value **49** has the meaning "ANY Mix Bus Channel". The integers that follow identify the mic/line audio input channels that are to be monitored. |
| Monitor programmable input state | A single integer in the range 1 to 30. This identifies which programmable input is monitored by the function. |
| Monitor active preset | A sequence of zero or more integers in the range 1 to 24. These identify which presets are to be monitored by the function. |
| Monitor hardware status | No target values necessary, the programmable output will become active if a fan fault or over-temperature condition is detected, or the real time clock battery is low. |
| Monitor virtual programmable input state | A single integer in the range 1 to 30. This identifies which virtual programmable input is monitored by the function. *Available in firmware versions 1.4.0 and higher.* |

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **prgoutdef(8)?<CR>** | **OK {3,1,3,4}<CRLF>** (Function "Monitor active preset" is assigned to programmable output 9. The presets being monitored are 1, 3, and 4.) |
| QUERY | **prgoutdef(1)?<CR>** | **OK {2,3}<CRLF>** (Function "Monitor programmable input state" is assigned to programmable output 1. The logical state of programmable input 3 is being monitored.) |
| QUERY | **prgoutdef(3)?<CR>** | **OK {1,3,1,7,8}<CRLF>** (Function "Monitor audio input activity" is assigned to programmable input 3. The mix bus being referenced is channel 3, and the audio inputs being monitored for activity are channels 1, 7, and 8.) |
| UPDATE | **prgoutdef(1)={2,5}<CR>** (Function "Monitor programmable input state" is assigned to programmable output 1. The programmable input being monitored is specified as number 5.) | **OK<CRLF>** |
| UPDATE | **prgoutdef(2)={3,1,2,4}<CR>** (Function "Monitor active preset" is assigned to programmable output 2. The presets to be monitored are specified as 1, 2, and 4.) | **OK<CRLF>** |
| UPDATE | **prgoutdef(4)={0}<CR>** ("No function" is assigned to programmable output 4, so that it will become inactive.) | **OK<CRLF>** |
| UPDATE | **prgoutdef(1)={1,5,3,4}<CR>** (Function "Monitor audio input activity" is assigned to programmable output 1. The mix bus being referenced is channel 5, and the audio inputs to be monitored are channels 3 and 4. | **OK<CRLF>** |

# prgoutht (programmable output channel activity hold time)

This command may be used as a query to read the hold time, or as an update to set it. The data type is integer, in the range 1 to 255, representing the hold time in one tenth second increments.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **prgoutht?<CR>** | **OK 4<CRLF>** |
| UPDATE | **prgoutht=10<CR>** | **OK<CRLF>** |

# prgoutiv (programmable output invert)

This command may be used as a query to read the programmable output invert status, or as an update to set the status. The programmable output is specified by using the address syntax. Addresses must be in the range 1 to 16. The data type is integer, either "1" meaning that the programmable output is inverted (contacts **open** when asserted), or "0" meaning that it is not (contacts **closed** when asserted, the normal case). If the address is wildcarded, then the data type is an array of integer of size 16. In this case the value **99** may be used in an update to indicate that a particular programmable output invert state is to **remain unchanged** by the command.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **prgoutiv(3)?<CR>** | **OK 0<CRLF>** |
| QUERY | **prgoutiv(*)?<CR>** | **OK {0,1,0,...,0,0,0}<CRLF>** |
| UPDATE | **prgoutiv(2)=1<CR>** | **OK<CRLF>** |
| UPDATE | **prgoutiv(*)={0,0,1,...,99,99,99}<CR>** | **OK<CRLF>** |

# prgoutpl (programmable output pulse)

This command may be used to generate one or more pulses on a programmable output. A single pulse may be generated by using the command form. The programmable output is specified by using the address syntax. Addresses must be in the range 1 to 16. More than one pulse may be generated with a single command by using the update form. In this case the data type is integer, with a value in the range 1 - 255. For each pulse, programmable output will change state for approximately 300 msec, then return to the original state. A train of consecutive pulses will have a 50% duty cycle. **Note:** a programmable output may be pulsed **only** if no function is assigned to it. If a function is assigned, the command will be ignored.

*The multiple pulse option is available in firmware versions 1.4.4 and higher.*

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **prgoutpl(5)<CR>** | **OK<CRLF>** |
| UPDATE | **prgoutpl(5)=3<CR>** | **OK<CRLF>** |

# prgoutqt (programmable output channel activity qualification time)

This command may be used as a query to read the qualification time, or as an update to set it. The data type is integer, in the range 1 to 255, representing the qualification time in one tenth second increments.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **prgoutqt?<CR>** | **OK 4<CRLF>** |
| UPDATE | **prgoutqt=10<CR>** | **OK<CRLF>** |

# vprgin (virtual programmable input state)

This command may be used to simulate a **momentary** contact closure on a virtual programmable input or as a query to read the state of virtual programmable inputs. The virtual programmable input is specified by using the address syntax. Addresses must be in the range 1 to 30. **Note:** simulated contact closures have no effect for progammable inputs configured for **analog gain control**. The data type returned by queries is integer, with the possible values depending on the nature of the function assigned to the programmable input. For **analog gain control** functions the values are the range 0 to 255, representing the voltage sensed by the programmable input (0 is 0 volts, 255 is 5 volts). For all other functions the value is either "1", meaning that the input is active (closed), or "0" meaning that it is not (open). The channel address may be wildcarded in queries, in which case the data type is an array of integer of size 30.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **vprgin(11)<CR>** | **OK<CRLF>** |
| QUERY | **vprgin(9)?<CR>** | **OK 1<CRLF>** |
| QUERY | **vprgin(*)?<CR>** | **OK {0,0,1,...,0,0,0}<CRLF>** |

# vprgindef (virtual programmable input definition)

This command may be used as a query to read a virtual programmable input definition, or as an update to modify the definition. The virtual programmable input is specified by using the address syntax. Addresses must be in the range 1 to 30. The data type is array of integer, with a variable length. The values contained in the array represent the definition of a virtual programmable input, using the following scheme:

- The **first** integer is a code that specifies the **function** that is assigned to the virtual programmable input. It may be in the range 0 to 19, with the following meanings:

| Code | Function | Code | Function |
|---|---|---|---|
| 0 | No function assigned | 13 | Momentary mute on crosspoint |
| 1 | Analog input gain control | 14 | Momentary unmute on input (PTT) |
| 2 | Increment input gain 1 dB | 15 | Analog crosspoint gain control |
| 3 | Decrement input gain 1 dB | 16 | Increment crosspoint gain 1 dB |
| 7 | Recall preset from memory | 17 | Decrement crosspoint gain 1 dB |
| 8 | Toggle mute on input | 18 | Run macros on close |
| 10 | Toggle mute on crosspoint | 19 | Run macros on close/open |
| 11 | Momentary mute on input | | |

- The choice of function determines what happens when the virtual programmable input is **asserted**. The choice of function affects the interpretation of the subsequent values in the array, which identify the **target** of the function.
  **NOTE: codes 4, 5, 6 and 9 are not used by the SPN32i.**
  **NOTE: Functions which control matrix crosspoints are limited to affecting a group of crosspoints that are members of a single row of the matrix. A row consists of all crosspoints associated with one particular input channel.**
  **NOTE: All gain changes made by programmable input functions affect rear panel gain only.**

- The **remaining** values specify the **target** of the function. These values are simply a list of one or more integers which identifies the thing(s) that are controlled by the function. The interpretation of the target values depends on the function that is assigned:

| Function | Target values |
|---|---|
| Analog input gain control | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Increment input gain 1 dB | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Decrement input gain 1 dB | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |

| | |
|---|---|
| Recall preset from memory | A single integer value in the range 1 to 24. This identifies which preset is to be recalled by the function. |
| Toggle mute on input | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Toggle mute on crosspoint | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Momentary mute on input | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Momentary mute on crosspoint | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Momentary unmute on input (PTT) | A sequence of zero or more integers in the range 1 to 36. These identify which input channels are to be controlled by the function. |
| Analog crosspoint gain control | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Increment crosspoint gain 1 dB | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Decrement crosspoint gain 1 dB | A single integer in the range 1 to 36, followed by a sequence of zero or more integers in the range 1 to 48. Together these values specify a group of crosspoints by identifying a single **input** channel and zero or more **mix bus** channels. |
| Run macro on close | A single integer value in the range 1 to 256. This identifies which macro is to be run by the function when the programmable input is asserted. |
| Run macros on close/open | Two integer values in the range 1 to 256. The first identifies which macro is to be run by the function when the programmable input is asserted. The second identifies which macro is to be run when the programmable input is subsequently de-asserted. |

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **vprgindef(9)?<CR>** | **OK {1,1,3,4,10}<CRLF>** (Function "Analog input gain control" is assigned to |

| | | |
|---|---|---|
| | | programmable input 9. The input channels under control are 1, 3, 4, and 10.) |
| QUERY | **vprgindef(2)?<CR>** | **OK {14,22}<CRLF>** (Function "Run a macro" is assigned to programmable input 2. Macro number 22 will be run if the programmable input is asserted.) |
| QUERY | **vprgindef(3)?<CR>** | **OK {13,3,1,7,8}<CRLF>** (Function "Momentary mute on crosspoint" is assigned to programmable input 3. The crosspoints under control are {3,1}, {3,7}, and {3,8}, which are members of the matrix row associated with input channel 3.) |
| QUERY | **vprgindef(4)?<CR>** | **OK {0}<CRLF>** (No function is assigned to programmable input 4.) |
| UPDATE | **vprgindef(11)={7,5}<CR>** (Function "Recall preset from memory" is assigned to programmable input 11. The preset to be recalled is specified as number 5.) | **OK<CRLF>** |
| UPDATE | **vprgindef(2)={8,1,2,4,5,6}<CR>** (Function "Toggle mute on input" is assigned to programmable input 2. The inputs to be controlled are specified as 1, 2, 4, 5, and 6.) | **OK<CRLF>** |
| UPDATE | **vprgindef(4)={0}<CR>** ("No function" is assigned to programmable input 4, so that asserting it will have no effect.) | **OK<CRLF>** |
| UPDATE | **vprgindef(1)={10,5,9,10}<CR>** (Function "Toggle mute on crosspoint" is assigned to programmable input 1. The crosspoints to be controlled are specified as {5,9} and {5,10}, which are members of the matrix row associated with input channel 5.) | **OK<CRLF>** |

# vprginundef (virtual programmable input un-definition)

This command may be used as an update to **un-define** one or more virtual programmable inputs, meaning that it no longer has any function assigned to it. The data type is array of integer, with a variable length. The values contained in the array represent the address of a virtual programmable input to be un-defined, in the range 1 to 30.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **vprginundef={3,4,5}<CR>**<br>(Virtual programmable inputs 3, 4, and 5 will be un-defined.) | **OK<CRLF>** |

# SPN32i Rear Panel Control Commands

| | |
|---|---|
| rpingn | Rear panel audio input gain |
| rpingnmin | Rear panel audio input gain minimum |
| rpingnpre | Rear panel audio input gain preset |
| rpingnr | Rear panel audio input gain ramp |
| rpingnst | Rear panel audio input gain step |
| rpsave | Save rear panel settings |
| rprest | Restore rear panel settings |
| rpxpgn | Rear panel matrix crosspoint gain |
| rpxpgnr | Rear panel matrix crosspoint gain ramp |
| rpxpgnst | Rear panel matrix crosspoint gain step |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## rpingn (rear panel audio input gain)

This command may be used as a query to read the rear panel input gain, or as an update to set the rear panel gain. The input channel or input group is specified by using the address syntax. Addresses for input channels must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for input groups must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher). The data type is integer, in the range -61 to 0, representing the gain in dB (the feature acts as an *attenuator*), where -61 has the special meaning "Off". The actual allowable range of values is determined by the gain minimum value. If the value sent in an update request is less than the gain minimum or greater than 0, it is ignored (no change is made). If the channel address is wildcarded, then the data type is an array of integer of size 36. In this case the value **99** may be used in an update to indicate that a particular rear panel input gain is to **remain unchanged** by the command. An input group is updated with a single

gain value which is applied to all members of the group. The response to a input group query or *verbose* update is an array of integer of size 36, containing the gain values for *all* input channels, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **rpingn(1)?<CR>** | **OK -3<CRLF>** |
| QUERY | **rpingn(101)?<CR>** | **OK {-3,-4,...,0,0}<CRLF>** |
| QUERY | **rpingn(*)?<CR>** | **OK {-3,-4,...,0,0}<CRLF>** |
| UPDATE | **rpingn(2)=0<CR>** | **OK<CRLF>** |
| UPDATE | **rpingn(101)=0<CR>** | **OK<CRLF>** |
| UPDATE | **rpingn(*)={0,-5,...,99,99}<CR>** | **OK<CRLF>** |

# rpingnmin (rear panel audio input gain minimum)

This command may be used as a query to read the rear panel input gain minimum, or as an update to set the gain minimum. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 36 (test signal inputs 33 - 36 are included). The data type is integer, in the range -61 to 0, representing the minimum gain in dB (i.e. the maximum *attenuation* possible using rear panel gain control). If the channel address is wildcarded, then the data type is an array of integer of size 36. The value -61 has the special meaning "Off", or infinite attenuation.

**Note:** The input gain minimum value also applies to *matrix crosspoint* rear panel gain control. Each value sets the gain minimum for all crosspoints in the crosspoint matrix *row* to which that input channel corresponds.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **rpingnmin(2)?<CR>** | **OK -30<CRLF>** |
| QUERY | **rpingnmin(*)?<CR>** | **OK {-30,-30,...,-30,-30}<CRLF>** |
| UPDATE | **rpingnmin(3)=-15<CR>** | **OK<CRLF>** |
| UPDATE | **rpingnmin(*)={-15,-15,...,-15,-15}<CR>** | **OK<CRLF>** |

# rpingnpre (rear panel audio input gain preset)

This command may be used as a query to read the rear panel input gain preset, or as an update to set the gain preset. The input channel is specified by using the address syntax. Addresses must be in the range 1 to 36 (test signal inputs 33 - 36 are included). The data type is integer, in the range -61 to 0, representing gain in dB. If the channel address is wildcarded, then the data type is an array of integer of size 36. The value -61 has the special meaning "Off", or infinite attenuation. For each input channel controlled by a contact closure (increment/decrement gain 1dB functions) this value is used to initialize the rear panel gain when the device is powered up.

**Note:** The input gain preset value also applies to *matrix crosspoint* rear panel gain control. Each value sets the gain preset for all crosspoints in the crosspoint matrix *row* to which that input channel corresponds. This setting has no effect for input channels or matrix crosspoints using analog (potentiometer) rear panel gain control.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **rpingnpre(3)?<CR>** | **OK -10<CRLF>** |
| QUERY | **rpingnpre(*)?<CR>** | **OK {-10,-10,...,0,0}<CRLF>** |
| UPDATE | **rpingnpre=-10<CR>** | **OK<CRLF>** |
| UPDATE | **rpingnpre(*)={-10,-10,...,0,0}<CR>** | **OK<CRLF>** |

# rpingnr (rear panel input gain ramp)

*This command is available in firmware versions 1.5.0 and higher.*

This command is used as an update to start a ramp of rear panel gain at the rate specified, either up or down. The input channel or input group is specified by using the address syntax. Addresses for input channels must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for input groups must be in the range 101 to 105 (there are 5 groups). The data type is array of integer, with a fixed length of 2. The values contained in the array represent the ramp *rate* and *gain limit* as follows:

- The **first** integer specifies the ramp *rate* in dB per second and must be in the range -6 to +6. A positive value indicates an upward gain ramp and a negative value indicates a downward gain ramp. The special value **0** stops a gain ramp in progress.
- The **second** integer specifies the ramp *gain limit* in dB, which is the point at which the ramp stops. It must be in the range -61 to 0,where -61 has the special meaning "Off". The actual allowable range of values is determined by the gain minimum value. If the value sent in an update request is less than the gain minimum value it is forced to the gain minimum value. If the current rear panel gain for the channel already equals or exceeds the gain limit then the command has no effect. *Not present if ramp rate is 0.*

The special array value {0} is used to stop a gain ramp currently in progress on an input channel or group of input channels.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **rpingnr(2)={-1,-30}<CR>**<br>**(start downward rp gain ramp on channel 2 at 1dB/sec rate, stop at -30dB)** | **OK<CRLF>** |
| UPDATE | **rpingnr(2)={0}<CR>**<br>**(stop rp gain ramp on channel 2)** | **OK<CRLF>** |
| UPDATE | **rpingnr(101)={2,0}<CR>**<br>**(start upward rp gain ramp on members of group 101 at 2dB/sec rate, stop at 0dB)** | **OK<CRLF>** |

# rpingnst (rear panel input gain step change)

This command is used as an update to step the gain by the amount specified, either up or down. The input channel or input group is specified by using the address syntax. Addresses for input channels must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for input groups must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher). The data type is integer, in the range -6 to +6, representing the gain step in dB. If the channel address is wildcarded, then the data type is an array of integer of size 36. A positive value increments the gain, a negative value decrements the gain. An input group is updated with a single gain step value which is applied to all members of the group. The response to a input group *verbose* update is an array of integer of size 36, containing the gain values for *all* input channels, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **rpingnst(2)=-2<CR>** | **OK<CRLF>** |
| UPDATE | **rpingnst(101)=-2<CR>** | **OK<CRLF>** |
| UPDATE | **rpingnst(*)={-2,-2,...,-2,-2}<CR>** | **OK<CRLF>** |

# rprest (restore rear panel settings)

This command may be used to restore the state of rear panel settings previously saved with the rpsave command. If used as a simple command, all rear panel settings are overwritten with the saved values. If used as an update, the data type is an integer whose value serves as a mask

which specifies which settings are restored and which are not. This mask has the same format as the preset mask.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **rprest<CR>** | **OK<CRLF>** |
| UPDATE | **rprest=1<CR>** | **OK<CRLF>** |

# rpsave (save rear panel settings)

This command may be used to save the current rear panel settings so that they can be restored at some future time by use of the rprest command. In this way a "snapshot" of the current rear panel gain and mute settings can be taken prior to some temporary reconfiguration of the device.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **rpsave<CR>** | **OK<CRLF>** |

# rpxpgn (rear panel crosspoint gain)

*This command is available in firmware versions 1.3.0 and higher.*

This command may be used as a query to read the rear panel crosspoint gain, or as an update to set the gain. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address.

In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire *column* of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire *row* of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`. Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded.

In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher).

The data type is integer, in the range -61 to 0, representing the gain in dB (the feature acts as an *attenuator*), where -61 has the special meaning "Off". If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. In these cases the value **99** may be used in an update to indicate that a particular crosspoint gain is to **remain unchanged** by the command. A crosspoint group is updated with a single gain value which is applied to all members of the group. The response to a crosspoint group query or *verbose* update is an array of size 48 if the group has been defined as a *Row* type, or an array of size 36 if it has been defined as a *Column* type. The array contains the gain values for *all* crosspoints in the row or column, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **rpxpgn(1,7)?<CR>** | **OK -3<CRLF>** |
| QUERY | **rpxpgn(101)?<CR>** | **OK {0,3,0,...0,0,0}<CRLF>** |
| QUERY | **rpxpgn(*,1)?<CR>** | **OK {0,3,0,...0,0,0}<CRLF>** |
| QUERY | **rpxpgn(1,*)?<CR>** | **OK {0,3,0,...0,0,0}<CRLF>** |
| QUERY | **rpxpgn(1:3,7)?<CR>** | **OK {0,3,0}<CRLF>** |
| QUERY | **rpxpgn(1,10:15)?<CR>** | **OK {0,3,0,0,0,0}<CRLF>** |
| UPDATE | **rpxpgn(5,2)=5<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgn(101)=5<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgn(*,5)={0,5,3,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgn(5,*)={0,5,3,...99,99,99}<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgn(1:3,7)={0,5,3}<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgn(1,10:15)={0,5,3,0,0,0}<CR>** | **OK<CRLF>** |

# rpxpgnr ( rear panel crosspoint gain ramp)

*This command is available in firmware versions 1.5.0 and higher.*

This command is used as an update to start a ramp of rear panel gain at the rate specified, either up or down. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address. In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48. In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups).

The data type is array of integer, with a fixed length of 2. The values contained in the array represent the ramp *rate* and *gain limit* as follows:

- The **first** integer specifies the ramp *rate* in dB per second and must be in the range -6 to +6. A positive value indicates an upward gain ramp and a negative value indicates a downward gain ramp. The special value **0** stops a gain ramp in progress.
- The **second** integer specifies the ramp *gain limit* in dB, which is the point at which the ramp stops. It must be in the range -61 to 0,where -61 has the special meaning "Off". The actual allowable range of values is determined by the gain minimum value. If the value sent in an update request is less than the gain minimum value it is forced to the gain minimum value. If the current rear panel gain for the channel already equals or exceeds the gain limit then the command has no effect. *Not present if ramp rate is 0.*

The special array value {0} is used to stop a gain ramp currently in progress on a crosspoint or group of crosspoints.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **rpxpgnr(2,3)={-1,-30}<CR>**<br>**(start downward rp gain ramp on crosspoint 2,3 at 1dB/sec rate, stop at -30dB)** | **OK<CRLF>** |
| UPDATE | **rpxpgnr(2,3)={0}<CR>**<br>**(stop rp gain ramp on crosspoint 2,3)** | **OK<CRLF>** |
| UPDATE | **rpxpgnr(101)={2,0}<CR>**<br>**(start upward rp gain ramp on members of group 101 at 2dB/sec rate, stop at 0dB)** | **OK<CRLF>** |

# rpxpgnst ( rear panel crosspoint gain step change)

*This command is available in firmware versions 1.3.0 and higher.*

This command is used as an update to step the rear panel crosspoint gain by the amount specified, either up or down. The crosspoint is specified by using the 2 dimensional address syntax, or a crosspoint group can be specified using a 1 dimensional address.

In the 2 dimensional case, addresses for the first dimension (input channel) must be in the range 1 to 36 (test signal inputs 33 - 36 are included). Addresses for the second dimension (mix bus) must be in the range 1 to 48, or wildcarded. If the input is wildcarded, then the command applies to the entire *column* of the matrix associated with the specified mix bus. If the output is wildcarded, then the command applies to the entire *row* of the matrix associated with the specified input channel. It is also possible to specify a subset of either the first dimension (input channels) or the second dimension (mix buses), **but not both**, using the range operator **':'** (colon character). For example, address `(3:5,7)` refers to 3 crosspoints: `(3,7)`, `(4,7)` and `(5,7)`.

Address `(5,7:10)` refers to 4 crosspoints: `(5,7)`, `(5,8)`, `(5,9)` and `(5,10)`. **Note:** when one dimension is specified as a range, the other may **not** be wildcarded.

In the 1 dimensional case, addresses must be in the range 101 to 105 (there are 5 groups, available in firmware versions 1.5.0 and higher).

The data type is integer, in the range -6 to +6, representing the gain step in dB. A positive value increments the gain, a negative value decrements the gain. If an entire column is being transmitted then the data type is array of integer of size 36. If an entire row is being transmitted then the data type is array of integer of size 48. A crosspoint group is updated with a single gain step value which is applied to all members of the group. The response to a crosspoint group *verbose* update is an array of size 48 if the group has been defined as a *Row* type, or an array of size 36 if it has been defined as a *Column* type. The array contains the gain values for *all* crosspoints in the row or column, not just those in the group.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **rpxpgnst(2,2)=2<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgnst(101)=2<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgnst(*,3)={2,2,2,...0,0,0}<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgnst(3,*)={-1,-1,-1,...0,0,0}<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgnst(3:8,12)={-1,-1,-1,-1,-1,-1}<CR>** | **OK<CRLF>** |
| UPDATE | **rpxpgnst(3,5:8)={-1,-1,-1,-1}<CR>** | **OK<CRLF>** |

# SPN32i Macro Management & Related Commands

| | |
|---|---|
| exit | Exit from macro |
| macro | Macro command |
| macroclr | Macro clear |
| macroti | Macro title |
| macvrport | Macro verbose response port |
| ropmac | "Run on Powerup" macro |
| run | Run a macro |
| sendcmd | Send command to ASPEN device |
| sendstr | Send string to port |
| sleep | Suspend macro execution |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## exit (exit a macro)

This command may be used to exit a **macro**, usually from within a conditional (if-then-else) statement within the macro.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **`exit<CR>`** | **`OK<CRLF>`** |

# macro (macro command)

This command may be used as a query to read one command from a macro, or as an update to set a command. The command is specified by using the 2 dimensional address syntax. Addresses for the first dimension specify the macro and must be in the range 1 to 256. Addresses for the second dimension specify the index of the command within the macro and must be in the range 1 to 64. The data type is string, with a limit of 110 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `macro` command to read or write a ASPEN command that already contains double-quote characters, for example the command `desc="whatever"`, which contains the quoted string argument `"whatever"`. The solution is to **escape** the double quotes within `desc="whatever"` so that it can itself be passed as a string argument for the `macro` command. This is done by preceding the double-quote characters with a **backslash** character like this: `desc=\"whatever\"`. Now it can be passed as a string argument to the `macro` command: `macro(1,1)="desc=\"whatever\""`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **macro(1,3)?<CR>** | **OK "ingn(3)=55"<CRLF>** |
| QUERY | **macro(1,4)?<CR>** | **OK "desc=\"Unit #1 East\""<CRLF>** |
| UPDATE | **macro(12,50)="xpmt(2,10)=1"<CR>** | **OK<CRLF>** |
| UPDATE | **macro(12,51)="desc=\"Classroom 17\""<CR>** | **OK<CRLF>** |

# macroclr (macro clear)

This command may be used to clear a macro. All lines in the macro will be erased. The macro is specified by using the address syntax. Addresses must be in the range 1 to 256.

Example:

| | REQUEST | RESPONSE |
|---|---|---|

| | | |
|---|---|---|
| COMMAND | **macroclr(3)<CR>** | **OK<CRLF>** |

# macroti (macro title)

This command may be used as a query to read the title of a macro, or as an update to set the title. The macro is specified by using the address syntax. Addresses must be in the range 1 to 256. The data type is string, with a limit of 30 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `macroti` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `macroti` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `macroti` command: `macroti(1,1)="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **macroti(1)?<CR>** | **OK "Sidebar nbr 2"<CRLF>** |
| UPDATE | **macroti(12)="Setup #3 West"<CR>** | **OK<CRLF>** |

# macvrport (macro verbose response port)

This command may be used as a query to determine the default port for verbose responses generated by macro execution, or as an update to set the port. The data is an integer type with the following possible values:

- **1** - RS232 port
- **2** - TCP port 1
- **3** - TCP port 2 (*firmware versions 1.4.0 and higher*)

When commands are executed within a macro, the response to the command is normally discarded. However, if a command is prefixed with an exclamation point (bang) character (verbose mode), the response is sent to either the RS232 port or the TCP port to provide

feedback to 3rd party control systems connected to these ports. This command is used to control which port receives the responses to verbose mode commands contained in a macro.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **macvrport?<CR>** | **OK 1<CRLF>** |
| UPDATE | **macvrport=2<CR>** | **OK<CRLF>** |

# ropmac ("run on powerup" macro)

This command may be used as a query to determine the "run on powerup" macro for the SPN32i. It may also be used as an update to set the macro. The data is an integer type in the range 0 to 256, where "0" has the special meaning "none".

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **ropmac?<CR>** | **OK 0<CRLF>** |
| UPDATE | **ropmac=5<CR>** | **OK<CRLF>** |

# run (run a macro)

This command may be used to run a macro. A single macro may be run by using the command form. In this case the macro is specified by using the address syntax. Addresses must be in the range 1 to 256. More than one macro may be run with a single command by using the update form. In this case the data type is array of integer, with a variable length in the range 1 - 16. The values contained in the array specify which macros to run.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **run(3)<CR>** | **OK<CRLF>** |
| UPDATE | **run={1,3,5}<CR>** | **OK<CRLF>** |

# sendcmd (send command to ASPEN device)

This command may be used as an update to allow one ASPEN device in a multi-unit system to send a command to another device in the system. There are two possibilities:

- `sendcmd` is executed on the master device in a multi-unit system to send a command to one or all slave devices
- `sendcmd` is executed on a slave device in a multi-unit system to send a command to the master device

The target ASPEN device is specified by using the address syntax. It is a relative address, in the range 1 to N where N is the number of ASPEN devices in the system reported by the `aspencnt` command. The address may be wildcarded if `sendcmd` is executed on the master device, causing all slave devices in the system to receive the command. If executed on a slave device, the address must be **(1)**, indicating the master device, or an ERROR response results. The command to be executed by the target device is given as the argument to the `sendcmd` command. The data type is string, with a limit of 127 characters in a quoted string argument, and 255 characters if the argument is supplied as a string variable reference (in macros only). Multiple commands can be combined into a semi-colon delimited list for efficient execution.

This command is intended for use within macros, and the command which is sent executes silently on the target device with no response returned to the sender. For this reason it makes sense to use it to send only action and update commands to the target device, not queries. When used in a macro, the argument may be a *string variable* reference containing the command(s) to be remotely executed. This can be convenient for "dynamically" building the command (or semi-colon delimited list of commands) using the format operator, string concatenation and other features of the macro language. Another advantage is that string variables can contain up to 255 characters. `sendcmd` may also be sent over a communications port by an external controller, but in this case the argument *must* be a quoted string..

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `sendcmd` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `macroti` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `macroti` command: `macroti(1,1)="The \"Hula\" Room"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **`sendcmd(1)="run(3)\r"<CR>`** (slave to master, quoted string | **`OK<CRLF>`** |

| | | |
|---|---|---|
| | argument) | |
| UPDATE | **sendcmd(1)="@foo@=5;@bar@=0;run(3)\r"<CR>** (slave to master, list of commands in quoted string argument) | **OK<CRLF>** |
| UPDATE | **sendcmd(3)="recall(5)\r"<CR>** (master to slave at relative address 3, quoted string argument) | **OK<CRLF>** |
| UPDATE | **sendcmd(*)="recall(5)\r"<CR>** (master to all slaves, quoted string argument) | **OK<CRLF>** |
| UPDATE | **sendcmd(*)=@cmd1@<CR>** (master to all slaves, variable argument , works *only* in a macro) | **OK<CRLF>** |

# sendstr (send string to port)

This command may be used as an update to send an arbitrary ASCII string to either the RS232 port or one of the TCP ports of an ASPEN device. The port, and optionally, the target ASPEN device, are specified by using the address syntax. The port address may be one of the following:

- **1** - RS232 port
- **2** - TCP port 1
- **3** - TCP port 2 (*firmware versions 1.4.0 and higher*)

The optional ASPEN address may be used to specify that the string be sent from some other device in a multi-unit system rather than locally. This allows system-wide access to a communications port located on some particular device. It is a relative address, in the range 1 to N where N is the number of ASPEN devices in the system reported by the `aspencnt` command. It may not be wildcarded. If `sendstr` is run on the master device in a multi-unit system any slave device may be specified, but if run on a slave device *only* the master device may be specified at address **(1)**. The ASCII characters to be sent are given as the argument to the `sendstr` command. The data type is string, with a limit of 127 characters in a quoted string argument, and 255 characters if the argument is supplied as a string variable reference (in macros only).

This command is intended for use within macros and is useful for sending strings to 3rd party equipment attached to the RS232 or TCP ports for control or notification purposes. It may also be sent over a communications port by an external controller, but in this case the argument *must* be a quoted string..

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `sendstr` command to read or write a string that already contains double-quote characters, for example: `The "Hula" Room`. The solution is to **escape** the double quotes within `The "Hula" Room` so that it can be passed as a string argument for the `macroti` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Hula\" Room`. Now it can be passed as a string argument to the `macroti` command: `macroti(1,1)="The \"Hula\" Room"`. Since the

**backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| UPDATE | **`sendstr(1)="FI22;"<CR>`** (local RS232 port, quoted string argument) | **`OK<CRLF>`** |
| UPDATE | **`sendstr(2,3)="FI22;"<CR>`** (master to slave at relative address 3, slave's TCP port 1, quoted string argument) | **`OK<CRLF>`** |
| UPDATE | **`sendstr(1,1)=@temp@`** (slave to master, master's RS232 port, variable argument, works *only* in a macro) | **`OK<CRLF>`** |

# sleep (suspend Macro execution)

*This command is available in firmware versions 1.4.4 and higher.*

This command may be used within a **macro** to suspend execution of the macro for a specified period of time. The time is expressed in milliseconds and is specified using the address syntax. The sleep time value is rounded off to a multiple of the effective minimum sleep time of approximately 250 msec.

**Note:** this command is effective only in *locally executed* macros. If it is contained in a list of semi-colon delimited commands sent to a remote unit using sendcmd, it will behave like exit and terminate execution. If sent over a communications port from an external controller, execution of macros in the device will be delayed until the sleep time has passed.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **`sleep(500)<CR>`** | **`OK<CRLF>`** |

# SPN32i Preset Management Commands

| | |
|---|---|
| actpre | Active preset number |
| default | Set a memory preset to factory defaults |
| defpre | Location of the memory preset recalled on powerup |
| predesc | Preset description |
| premsk | Default preset mask |
| preror | Preset "Run on Recall" Macro |
| recall | Recall a memory preset |
| store | Store settings to a memory preset location |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## actpre (active preset)

This command may be used as a query to determine which memory preset is active (that is to say, which preset was last recalled from memory). It may also be used as an update to set the active preset by recalling the specified preset. The data is an integer type in the range 1 to 24.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`actpre?<CR>`** | **`OK 12<CRLF>`** |
| UPDATE | **`actpre=4<CR>`** | **`OK<CRLF>`** |

# default (default settings)

This command may be used to restore a memory preset to the factory defaults. The preset location to be modified is specified by using the address syntax. Addresses must be in the range 1 to 24. No data is transferred.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **default(3)<CR>** | **OK<CRLF>** |

# defpre (default preset)

This command may be used as a query to determine the memory preset location used as the powerup default. It may also be used as an update to set the default preset. The data is an integer type in the range 0 to 24, where 0 has the special meaning "On powerup, the preset that was active when the unit was last powered down will be recalled". This provides a "memory" capability useful for some applications, and is called the **Last Preset** option.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **defpre?<CR>** | **OK 11<CRLF>** |
| UPDATE | **defpre=2<CR>** | **OK<CRLF>** |

# predesc (preset description)

This command may be used as a query to read the user defined preset description for the **active preset**. It may also be used as an update to set the description. The data is a string type, with a limit of 60 characters.

**Note:** String arguments in commands need to be passed in **quoted** form, contained in a pair of **double-quote** (") characters. A problem arises when using the `predesc` command to read or write a string that already contains double-quote characters, for example: `The "Basic" setup`. The solution is to **escape** the double quotes within `The "Basic" setup` so that it can be passed as a string argument for the `desc` command. This is done by preceding the double-quote characters with a **backslash** character like this: `The \"Basic\" setup`. Now it can be passed as a string argument to the `desc` command: `desc="The \"Basic\" setup"`. Since the **backslash** serves as the escape character in quoted-string arguments, it too must be escaped if it is part of

the string, so `"foo\bar"` would become `"foo\\bar"`. If necessary, any character, printable or non-printable, can be represented in the hexadecimal escaped form `\xHH` where `HH` is any 2-digit hexadecimal number. The special escaped character forms `\r` (carriage return), `\n` (new line) and `\t` (tab) are also recognized.

**Important:** changes to this setting do not become permanent until the preset is stored to nonvolatile memory!

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **predesc?<CR>** | **OK "Wedding #1, no hallway speakers"<CRLF>** |
| UPDATE | **predesc="Weekly Rotary Club breakfast meeting."<CR>** | **OK<CRLF>** |

# premsk (default preset mask)

This command may be used as a query to determine the default preset mask in effect for the **active preset**. It may also be used as an update to set the default mask. The preset mask is a number which determines whether or not certain mute and rear panel gain settings are preserved or overwritten when a preset recall occurs. The default mask is in effect unless overridden by the recall command. The data is an integer type formed by adding together one or more of the following values:

- **1** to preserve input channel rear panel gain settings
- **2** to preserve input channel mute settings
- **16** to preserve the state of programmable inputs
- **32** to preserve programmable input function definitions
- **64** to preserve matrix crosspoint rear panel gain settings
- **128** to preserve matrix crosspoint mute settings

**NOTE:** values 4 and 8 are not used by the SPN32i.

For instance, to preserve input and output mutes on preset recalls, but not the rear panel gains, the mask value would be calculated as 2 + 8 = 10. A value of 0 means that the preset recall is "hard", with all mutes and rear panel gains overwritten by the values contained in the newly active preset.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **premsk?<CR>** | **OK 2<CRLF>** |
| UPDATE | **premsk=3<CR>** | **OK<CRLF>** |

# preror (preset "run on recall" macro)

This command may be used as a query to determine the "run on recall" macro for the **active preset** . It may also be used as an update to set the macro. The data is an integer type in the range 0 to 256, where "0" has the special meaning "none".

**Important:** changes to this setting do not become permanent until the preset is stored to nonvolatile memory!

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **preror?<CR>** | **OK 0<CRLF>** |
| UPDATE | **preror=5<CR>** | **OK<CRLF>** |

# recall (recall preset)

This command may be used to recall a memory preset, making it the active preset. The preset location to be recalled is specified by using the address syntax. Addresses must be in the range 1 to 24. If used as a simple command, no data is transferred, and the preset recall is controlled by the default preset mask. If used as an update, the data type is an integer whose value serves as the preset mask for the recall operation, overriding the default preset mask. See premsk.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **recall(3)<CR>** | **OK<CRLF>** |
| UPDATE | **recall(4)=1<CR>** | **OK<CRLF>** |

# store (store settings to preset)

This command may be used to store current settings to a memory preset. The preset location to be updated is specified by using the address syntax. Addresses must be in the range 1 to 24. No data is transferred.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| COMMAND | **store(3)<CR>** | **OK<CRLF>** |

# SPN32i Signal Generator Commands

| | |
|---|---|
| pnlv | Pink Noise level |
| swplv | Swept Sine level |
| swpen | Swept Sine sweep enable |
| swpset | Swept Sine generator settings |
| tonefrq | Test Tone frequency |
| tonelv | Test Tone level |
| wnlv | White Noise level |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## pnlv (pink noise level)

This command may be used as a query to read the Pink Noise generator output level, or as an update to set it. The data type is integer, in the range -70 to +20, representing the level in dBu.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`pnlv?<CR>`** | **`OK 0<CRLF>`** |
| UPDATE | **`pnlv=-10<CR>`** | **`OK<CRLF>`** |

## swpen (swept sine sweep enable)

This command may be used as a query to read the Swept Sine generator sweep enable status, or as an update to set it. The data type is integer, either "1" meaning that the sweep is enabled (started), or "0" meaning that it is not (stopped).

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **swpen?<CR>** | **OK 0<CRLF>** |
| UPDATE | **swpen=1<CR>** | **OK<CRLF>** |

# swplv (swept sine level)

This command may be used as a query to read the Swept Sine generator output level, or as an update to set it. The data type is integer, in the range -70 to +20, representing the level in dBu.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **swplv?<CR>** | **OK 0<CRLF>** |
| UPDATE | **swplv=-10<CR>** | **OK<CRLF>** |

# swpset (swept sine generator settings)

This command may be used as a query to read the Swept Sine generator settings, or as an update to modify the settings. The data type is array of integer, with a fixed length of 4. The values contained in the array represent the settings, using the following scheme:

- The **first** integer is a code that specifies the **mode and wave shape** which specifies:
  - **Sweep mode** - Whether the generator is to sweep the sine wave frequency continuously after starting, or just once (single sweep).
  - **Sweep shape** - Whether the sweep proceeds **up** in frequency from start frequency to stop frequency and then **down** in frequency from there back to the start frequency (triangle) **or** proceeds **up** in frequency from start frequency to stop frequency, then reverts back to the start frequency and sweeps up again. (sawtooth).
  - **Sweep rate** - Whether the frequency is swept in time at a linear rate (constant number of Hz per second), or a logarithmic rate (constant number of octaves per second).

The code may be in the range 0 to 7, with the following meanings:

| Code | Function | Code | Function |
|---|---|---|---|
| 0 | Single sweep, sawtooth, linear | 4 | Single sweep, sawtooth, logarithmic |
| 1 | Continuous sweep, sawtooth, linear | 5 | Continuous sweep, sawtooth, logarithmic |
| 2 | Single sweep, triangle, linear | 6 | Single sweep, triangle, logarithmic |
| 3 | Continuous sweep, triangle, linear | 7 | Continuous sweep, triangle, logarithmic |

- The **second** integer specifies the start frequency of the sweep. It may be in the range 20 to 20000, and represents frequency in Hz.
- The **third** integer specifies the stop frequency of the sweep. It may be in the range 20 to 20000, and represents frequency in Hz.
- The **fourth** integer specifies the sweep time. It may be in the range 10 to 600, and represents time in 1/10 second increments.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **swpset?<CR>** | **OK {4,20,10000,100}<CRLF>** |
| UPDATE | **swpset={5,20,10000,100}<CR>** | **OK<CRLF>** |

# tonefrq (test tone frequency)

This command may be used as a query to read the Test Tone generator output frequency, or as an update to set it. The data type is integer, in the range 20 to 20000, representing the frequency in Hz.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **tonefrq?<CR>** | **OK 440<CRLF>** |
| UPDATE | **tonefrq=1000<CR>** | **OK<CRLF>** |

# tonelv (test tone level)

This command may be used as a query to read the Test Tone generator output level, or as an update to set it. The data type is integer, in the range -70 to +20, representing the level in dBu.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **tonelv?<CR>** | **OK 0<CRLF>** |
| UPDATE | **tonelv=-10<CR>** | **OK<CRLF>** |

# wnlv (white noise level)

This command may be used as a query to read the White Noise generator output level, or as an update to set it. The data type is integer, in the range -70 to +20, representing the level in dBu.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **wnlv?<CR>** | **OK 0<CRLF>** |
| UPDATE | **wnlv=-10<CR>** | **OK<CRLF>** |

# SPN32i Real Time Clock, Timer & Alarm Commands

| | |
|---|---|
| alarm | Alarm definition |
| alarmen | Alarm enable |
| date | Date settings |
| time | Time settings |
| timer | Timer definition |
| timeren | Timer enable |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## alarm (alarm definition)

This command may be used as a query to read an alarm definition, or as an update to set it. The alarm is specified by using the address syntax. Addresses must be in the range 1 to 4. The data type is array of integer, with a fixed length of 6. The values contained in the array represent the settings, using the following scheme:

- The **first** integer is formed by adding together one or more of the following values:
  - **1** - Sunday
  - **2** - Monday
  - **4** - Tuesday
  - **8** - Wednesday
  - **16** - Thursday
  - **32** - Friday
  - **64** - Saturday

For example, to set the alarm to expire on Tuesday and Thursday of each week, the value is **20** (4 + 16).

- The **second** integer specifies the hour of the day at which the alarm expires in 24 hour format. It may be in the range 0 to 23.
- The **third** integer specifies the minute of the hour at which the alarm expires. It may be in the range 0 to 59.
- The **fourth** integer specifies the macro which will be run when the alarm expires. It may be in the range 1 to 256.
- The **fifth** integer specifies whether or not the alarm is repeating. It may be either "1" meaning that the alarm will be re-enabled ( re-armed) after it expires, or "0" meaning that it will not.
- The **sixth** integer specifes the enable status of the alarm. It may be either "1" meaning that it is enabled (armed) or "0" meaning that it is not.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **alarm(2)?<CR>** | **OK {1,8,30,20,1,1}<CRLF>** |
| UPDATE | **alarm(2)={1,8,30,20,1,1}<CR>** | **OK<CRLF>** |

# alarmen (alarm enable)

This command may be used as a query to read the alarm enable status, or as an update to set it. The data type is integer, either "1" meaning that the alarm is enabled (armed), or "0" meaning that it is not.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **alarmen(1)?<CR>** | **OK 0<CRLF>** |
| UPDATE | **alarmen(2)=1<CR>** | **OK<CRLF>** |

# date (date settings)

This command may be used as a query to read the date settings, or as an update to modify the settings. The data type is array of integer, with a fixed length of 4. The values contained in the array represent the settings, using the following scheme:

- The **first** integer specifies the year of the century. It may be in the range 0 to 99.
- The **second** integer specifies the month of the year. It may be in the range 1 to 12.
- The **third** integer specifies the date of the month. It may be in the range 1 to 31.
- The **fourth** integer specifies the day of the week. It may be in the range 1 to 7.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **date?<CR>** | **OK {9,8,7,5}<CRLF>** |
| UPDATE | **date={9,8,7,5}<CR>** | **OK<CRLF>** |

# time (time settings)

This command may be used as a query to read the time settings, or as an update to modify the settings. The data type is array of integer, with a fixed length of 4. The values contained in the array represent the settings, using the following scheme:

- The **first** integer specifies the hour, in 24 hour format. It may be in the range 0 to 23.
- The **second** integer specifies the minute. It may be in the range 0 to 59.
- The **third** integer specifies the second. It may be in the range 0 to 59.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **time?<CR>** | **OK {14,20,32}<CRLF>** |
| UPDATE | **time={20,5,0}<CR>** | **OK<CRLF>** |

# timer (timer settings)

This command may be used as a query to read a timer definition, or as an update to set it. The timer is specified by using the address syntax. Addresses must be in the range 1 to 4. The data type is array of integer, with a fixed length of 6. The values contained in the array represent the settings, using the following scheme:

- The **first** integer specifies the number of hours in the timer period in 24 hour format. It may be in the range 0 to 23.
- The **second** integer specifies the number of minutes in the timer period. It may be in the range 0 to 59.

- The **third** integer specifies the number of seconds in the timer period. It may be in the range 0 to 59.
- The **fourth** integer specifies the macro which will be run when the timer expires. It may be in the range 1 to 256.
- The **fifth** integer specifies whether or not the timer is repeating. It may be either "1" meaning that the timer will be re-enabled ( re-armed) after it expires, or "0" meaning that it will not.
- The **sixth** integer specifes the enable status of the timer. It may be either "1" meaning that it is enabled (armed) or "0" meaning that it is not.

The maximum timer period is 23hr:59m:59s, the minimum period is 1 second.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **timer(3)?<CR>** | **OK {0,30,0,120,0,1}<CRLF>** |
| UPDATE | **timer(3)={0,30,0,120,0,1}<CR>** | **OK<CRLF>** |

# timeren (timer enable)

This command may be used as a query to read the timer enable status, or as an update to set it. The data type is integer, either "1" meaning that the timer is enabled (armed), or "0" meaning that it is not.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **timeren(3)?<CR>** | **OK 0<CRLF>** |
| UPDATE | **timeren(3)=1<CR>** | **OK<CRLF>** |

# SPN32i Event Commands

| | |
|---|---|
| eventmac | Event macro |
| eventen | Event enable status |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## eventmac (event macro)

This command may be used as a query to read an Event channel macro number, or as an update to set the macro number. The event channel is specified by using the address syntax. Addresses must be in the range 1 to 5, and specify of of the following events:

- **1** - Rear Panel input gain change
- **2** - Rear Panel input mute control change
- **3** - Rear Panel crosspoint gain change
- **4** - Rear Panel crosspoint mute change
- **5** - Hardware status alert
- **6** - Input channel activity change (*firmware versions 1.4.0 and higher*)

The data type is integer, in the range 1 to 256, representing the number of the macro to be run if the event is raised.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`eventmac(1)?<CR>`** | **`OK 25<CRLF>`** |
| UPDATE | **`eventmac(5)=20<CR>`** | **`OK<CRLF>`** |

# eventen (event enable status)

This command may be used as a query to read an Event channel enable status, or as an update to set the status. The event channel is specified by using the address syntax. Addresses must be in the range 1 to 5, and specify of of the following events:

- **1** - Rear Panel input gain change
- **2** - Rear Panel input mute control change
- **3** - Rear Panel crosspoint gain change
- **4** - Rear Panel crosspoint mute change
- **5** - Hardware status alert
- **6** - Input channel activity change (*firmware versions 1.4.0 and higher*)

The data type is integer, either "1" meaning that the event channel is enabled, or "0" meaning that it is disabled.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **eventen(3)?<CR>** | **OK 0<CRLF>** |
| UPDATE | **eventen(2)=1<CR>** | **OK<CRLF>** |

# SPN32i Network Setup Commands

| | |
|---|---|
| defgate | Default gateway |
| dhcpen | DHCP enable |
| httpport | HTTP port number |
| ipaddr | IP address |
| macaddr | MAC address |
| netmask | Network mask |
| tcpport | TCP port number |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## defgate (default gateway)

This command may be used as a query to read the Default Gateway address, or as an update to set it. The data type is string, containing the address in IP "dotted quad" format.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`defgate?<CR>`** | **`OK "172.16.4.1"<CRLF>`** |
| UPDATE | **`defgate="172.16.4.1"<CR>`** | **`OK<CRLF>`** |

## dhcpen (DHCP enable)

This command may be used as a query to read the DHCP enable status, or as an update to set it. The data type is integer, either "1" meaning that the DHCP client feature is enabled, or "0" meaning that it is not. If enabled, DHCP (Dynamic Host Configuration Protocol) is used at power up to obtain an IP address. **Note:** If this setting is changed, the new value takes effect the next time the device is powered up.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **dhcpen?<CR>** | **OK 0<CRLF>** |
| UPDATE | **dhcpen=1<CR>** | **OK<CRLF>** |

# httpport (HTTP port number)

This command may be used as a query to read the HTTP port number assignment, or as an update to set it. The data type is integer, in the range 0 to 65535, representing the port number used for HTTP connections to the device. The default value is 80.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **httpport?<CR>** | **OK 80<CRLF>** |
| UPDATE | **httpport=80<CR>** | **OK<CRLF>** |

# ipaddr (IP address)

This command may be used as a query to read the IP address of the device, or as an update to set it. The data type is string, containing the address in IP "dotted quad" format.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **ipaddr?<CR>** | **OK "172.16.4.151"<CRLF>** |
| UPDATE | **ipaddr="172.16.4.151"<CR>** | **OK<CRLF>** |

# macaddr (MAC address)

This command may be used as a query to read the ethernet MAC address of the device. The data type is string, containing the address in IEEE MAC-48 format.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **macaddr?<CR>** | **OK "00-24-34-32-00-22"<CRLF>** |

## netmask (network mask)

This command may be used as a query to read the Network Mask, or as an update to set it. The data type is string, containing the mask in IP "dotted quad" format.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **netmask?<CR>** | **OK "255.255.255.0"<CRLF>** |
| UPDATE | **netmask="255.255.255.0"<CR>** | **OK<CRLF>** |

## tcpport (TCP port number)

This command may be used as a query to read the TCP port number assignment, or as an update to set it. The data type is integer, in the range 0 to 65535, representing the port number used for TCP connections to the device. The default value is 4080.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **tcpport?<CR>** | **OK 4080<CRLF>** |
| UPDATE | **tcpport=4080<CR>** | **OK<CRLF>** |

# SPN32i ASPEN System Commands

| | |
|---|---|
| aspen | ASPEN device query |
| aspenaddr | ASPEN default address |
| aspencnt | ASPEN device count |
| aspenum | ASPEN device enumerate |

**Termination:** all commands are terminated with an ASCII **carriage return** character (hex code 0x0D), represented by **<CR>** in the examples below. All responses are terminated with an ASCII **carriage return, line feed** pair (hex codes 0x0D, 0x0A), represented by **<CRLF>** in the examples below. An ellipsis (**...**) represents members of an array that have been omitted from an example for the sake of brevity.

**Verbose response:** commands prefixed with an exclamation point (bang) character result in a "verbose" response containing both the name of the property being addressed and its current value (if any). The verbose response returns the property/value pair in the "assignment" form, for example **`OK ingn(2)=40 <CRLF>`**. This supports certain 3rd party control programming styles where the response needs to be self-describing.

## aspen (ASPEN device query)

This command may be used as a query to read the id string and serial number of a device enumerated in an ASPEN system. This command is recognized only by the master unit in a multi-unit system. A relative ASPEN address is specified by using the address syntax. Addresses must be in the range 1 to N where N is the count of devices in the system discovered with the `aspencnt` command. The data type is string, and contains the serial number and device id joined by a colon (:) character.

Example:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **`aspen(2)?<CR>`** | **`OK "5000101:SPN1624"<CRLF>`** |

## aspenaddr (ASPEN default address)

This command may be used as a query to read the ASPEN default address, or as an update to set it. This command is recognized only by the master unit in a multi-unit system, and sets the default address for communication over a particular port (RS232, USB or TCP). The data type is string, and contains the serial number of the device to be the default command target. Once set, commands sent without an ASPEN address specified are routed to this default address. This command is used to designate a particular slave unit in the system as the default command target, rather than the master unit. This can be convenient for 3rd party control programs. To reset the default address back to that of the master unit, this command is sent with an empty address ("").

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **aspenaddr?<CR>** | **OK "5000150";CRLF>** |
| UPDATE | **aspenaddr="5000125"<CR>** | **OK<CRLF>** |

# aspencnt (ASPEN device count)

This command may be used as a query to read the count of ASPEN, devices in a system. This count is the result of the ASPEN system enumeration process, and can be used together with the aspen command to to discover the identities of the devices in the system. The data type is integer.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|
| QUERY | **aspencnt?<CR>** | **OK 3<CRLF>** |

# aspenum (ASPEN device enumerate)

This command may be used force an enumeration of the devices present in an ASPEN system. This command is recognized only by the master unit in a multi-unit system. Normally, the master device conducts an enumeration of the aspen devices in the system immediately after powerup, and every 5 minutes or so afterwords, but this command makes it possible to force an enumeration at any time. The enumeration process requires a minute or two to complete, after which time the results can be discovered by using the aspencnt and aspen commands.

Examples:

| | REQUEST | RESPONSE |
|---|---|---|

| | | |
|---|---|---|
| COMMAND | **aspenum<CR>** | **OK<CRLF>** |