LANGUAGE REFERENCE GUIDE

SNAPI
STANDARD NETLINX API

V1.15 - COMPONENTS/LISTENERS

AV FOR AN IT WORLD

Table of Contents

Interfacing with Standard NetLinx API (SNAPI)

Overview

The Standard NetLinx API (SNAPI) maps function and feedback calls in Duet modules to ICSP channels, levels and commands. SNAPI allows NetLinx programmers to utilize Duet modules in their NetLinx programs and access the function and feedback of those modules through programming similar to programming they would use on an AMX device, such as a volume box. While each Duet module may support advanced functions via channels, levels and commands (see specific module documentation for the channels, levels and commands supported by that module), the SNAPI mappings apply to the Standard API supported by each module.

Devices

In Duet, all modules use a Duet virtual device. Duet virtual devices are in the range 41000:1:0 to 42000:1:0. Duet virtual devices are specifically designed for use with Duet modules. Regular virtual devices will not work properly with Duet modules.

Ports

In Duet, each device may support one or more ports. Multiple ports are used to provide access to different components within the module. For instance, a switcher may support output volume for each switcher output. In Duet, this is represented by a volume component for each output and in SNAPI, each of these components is mapped to a Duet virtual device ports. Port 1 will control the volume for output 1, port 2 will control the volume for output 2, etc… Another common use of port is to control different "zones" of and Audio Receiver, HVAC, Security Systems, etc… It is possible that a single port may offer only a small bit of functionality, such as volume control.

Port 1 is always the "main" device and supports all functions of the device. For components that have multiple instances, such as volume, port 1 will control the default component, which is usually component 1.

The documentation for each module will describe what ports are available and what functions they control. See the specific module documentation for a complete list of ports supported by the module.

Channels

In SNAPI, there are four kinds of channels: Input Function Channels, Momentary Function Channels, Discrete Function Channels and Feedback Channels.

Input function channels are used for response type functions, such as when a device wants to inform your program of an interesting event, similar to a button push on a touch panel. For instance, channel 1 is the input function channel for processButtonStateEvent (). When the module has information about the push or release of a button, the module will send a PUSH or RELEASE.

Momentary function channels are used for momentary type functions and do not provide discrete feedback. For instance, channel 9 is the momentary function channel for cyclePower(). When pulsed, the channel cycles the state of power on the device and only provides momentary feedback, i.e. the channel is on only while this function is activated.

Discrete function channels are used for discrete type functions and usually provide discrete feedback. For instance, channel 255 is the discrete function channel for setPower(); When turned on, this channel sets the state of the power on the device to on. When turned off, this channel sets the state of the power on the device to off. This channel provides discrete feedback as well; this channel is on if and only if the state of the power on the device is on. In most cases, the channel is listed as a Discrete function channel next to the function the channel controls and as a Feedback channel next to the feedback function that controls this channel.

Feedback channels provide discrete feedback only. For instance, channel 251 is the feedback channel of Communication Active. This channel is on if and only if the module is able to communicate to the device.

Levels

Levels in SNAPI are used for both function and feedback. In some cases, a level is only used for feedback, such as temperature, while some levels are used for function and feedback, such a volume. In most cases, level ranges are from 0-255. All exceptions to this rule are noted, and are only used when the level range is not bounded by a minimum and a maximum, such as temperature.

Commands

Commands in SNAPI are used for discrete and momentary functions when the function requires textual information, multiple parameters, or the functions are not commonly used. For instance, Temperature scale is set via a command because this is usually done only once in a control system program.

Other functions, such as adding and removing lighting and keypad addresses, requires more information than a channel or level alone can convey. All commands start with a command header, followed by a "-" to separate the command from the data, and data arguments are usually separated by ","s.

Commands used to query for the status of a property start with a "?". Query commands cause the module to respond with a response command.

SNAPI.axi

SNAPI.axi is an include file that defines constants for each channel and level defined by SNAPI. These constants can be used in your programs in place of channel and level numbers. The constant names are listed in this document alongside every SNAPI function assigned to a channel or level.

SNAPI.axi is located in C:\Program Files\Common Files\AMXShare\AXIs. The file is organized by device type and lists all the standard channels and levels that may be supported by the module. This list does not contain all the channel or levels supported by a module and may include channels and levels not supported by a device. See the specific module documentation for a complete list of channels and levels supported by the module.

To include SNAPI.axi in your program, simply add an #INCLUDE statement for it:

```
#INCLUDE 'SNAPI.axi'
```

The file does not need to be copied to your project directory. The NetLinx compiler will be able to find this file automatically and include it in your program.

Programming

Channels

Input function channels are used for response type functions. For instance, channel 1 notifies your program of a button push in the KeypadComponent. When your program receives a PUSH, the button is pushed. When your program receives a RELEASE, the button is released. You should use BUTTON_EVENT's to capture the changes of an Input function channel:

```
BUTTON_EVENT[dvDevice, KEYPAD_BTN]
{
   PUSH:       // Button was pushed
   {
      {
      RELEASE: // Button was released
      }
   }
}
```

Momentary function channels are used to activate functions when the channels change from an OFF state to an ON state. For instance, channel 9 or the constant POWER cycles the state of the power on the device when it turns on. No change occurs when the channels change from an ON state to an OFF state. You should activate Momentary function channels using the PULSE, TO or MIN_TO keywords:

```
PULSE[dvDevice,POWER]    // Cycle the state of power
TO[dvDevice,POWER]       // Cycle the state of power
MIN_TO[dvDevice,POWER]   // Cycle the state of power
```

Discrete function channels are used to activate functions when the channel changes from an OFF state to an ON state and from an ON to an OFF state. For instance, channel 255 or the constant POWER_ON sets the state of the power on the device when it turns on and off. You should activate discrete function channels using the ON and OFF keywords, or any syntax that changes the state of the channel such as a feedback assignment:

```
ON[dvDevice,POWER_ON]                        // Turn the power on
OFF[dvDevice,POWER_ON]                       // Turn the power off
[dvDevice,POWER_ON] = ![dvDevice,POWER_ON]   // Cycle the state of power
```

Feedback channels do not active function and should only be used for feedback.

These channels can be used in CHANNEL_EVENTs or feedback assignment statements to read the status of the channel:

```
bCommunicationActive = [dvDevce,DEVICE_COMMUNICATING]
CHANNEL_EVENT[dvDevice,DEVICE_COMMUNICATING]
{
    ON:
        ON[bCommunicationActive]
    OFF:
        OFF[bCommunicationActive]
}
```

Ramping Channels

Some channels in SNAPI provide ramping functionality and some provide adjust "stepping" functionality. Since ramping on a device is only provided if the device supports ramping, a channel that causes ramping on one device may not cause ramping on another device. The following syntax can be used universally for all ramping functionality:

```
BUTTON_EVENT[dvTP,1]
{
    PUSH:
        TO[dvDevice,VOL_UP]
    HOLD[3 , REPEAT]:
        ON[dvDevce,VOL_UP]
}
```

The PUSH: TO part of the button event causes ramping to start and continue until the button is released. If the device does not support ramping, the device adjusts the desired parameter either up or down one step and stops. The HOLD: ON part of the button event causes the step adjustment to repeat, at a rate specified by the HOLD repeat time, until the button is released. The HOLD: ON part of the button event has no effect if the device supports ramping.

In a future version of Duet, it is expected that all modules will support ramping natively and that this NetLinx code will not always be required. However, if the module you are using does not support ramping, this code can be used to achieve ramping functionality.

Levels

Levels in SNAPI are used for both function and feedback. For feedback levels, the level value can be captured in a LEVEL_EVENT, with CREATE_LEVEL or sent directly to a touch panel display bargraph using DEFINE_CONNECT_LEVEL:

```
LEVEL_EVENT[dvDevice,1]
{
    // LEVEL.VALUE holds the new level value
}
CREATE_LEVEL dvDevice,1,nMyVariable // nMyVariable will hold the
                                    // latest value of the level
DEFINE_CONNECT_LEVEL(dvDevice,1,dvTp,1)
```

Levels used for functions can be set by calling SEND_LEVEL or by connecting to a touch panel active bargraph using DEFINE_CONNECT_LEVEL:

```
SEND_LEVEL dvDevice,1,nNewLevelValue
DEFINE_CONNECT_LEVEL(dvDevice,1,dvTp,1)
```

The CREATE_LEVEL/SEND_LEVEL mechanism is recommended for use with SNAPI. While LEVEL_EVENT will work fine, you may experience problems when a touch panel falls offline and then reconnects, which happens often with wireless panels. LEVEL_EVENT's will only fire when a change of the level value occurs. When the panel comes online, the only way to reliably update the level is with a SEND_LEVEL.

Loading Duet Modules

The following code example represents an alternative method that reduces the need to re-initialize the Duet Module, in order to reduce boot-time on the Master.

```
DEFINE_VARIABLE

VOLATILE CHAR DENON_AVR-3803_DUET_PROPERTIES[][] =
{
     // Standard module properties
     'Physical-Device=5001:1:0',
     'Duet-Device=41001:1:0',
     'Duet-Module=Denon_AVR-3803_dr1_0_0',
     'Bundle-Version=1.0.0',
     'Device-Category=ip,serial,rs-232',
     'Device-Make=Denon',
     'Device-Model=AVR-3803',
     'Device-SDKClass=com.amx.duet.devicesdk.Receiver',
     'Device-Revision=1.0.0'
     // Optional properties, refer to module documentation to determine
     // which properties are supported and for usage details
     // 'Baud_Rate'
     // 'Poll_Time'
     // 'Reconnect_Time'
     // 'Password'
     // 'User_Name'
     // 'IP_Address'
     // 'Port'
     // 'IP_Type'
     // 'Device_ID'
     // 'Timeout_Count'
     // 'System_Diagnostic'
}

(***********************************************************)
(*                STARTUP CODE GOES BELOW                  *)
(***********************************************************)
DEFINE_START

// Load Duet Module
LOAD_DUET_MODULE(DENON_AVR-3803_DUET_PROPERTIES)
```

Commands

Commands in SNAPI are sent like commands to other devices, using the SEND_COMMAND keyword:

```
SEND_COMMAND dvDevice,'?VERSION'
```

Commands used to query for the status of a property start with a "?". Query commands cause the module to respond with a response command. Note that this response is a command, not a string and can be captured in a DATA_EVENT in the COMMAND sub-section:

```
DATA_EVENT[dvDevice]
{
    COMMAND:
    {
        // DATA.TEXT holds the response to a query command
    }
}
```

General

The NetLinx program should assume that NetLinx levels are initially 0 and that channels are 'off'. The SNAPI router will notify the NetLinx client upon a change of state.

All Duet Virtual Devices should be created on port 1, e.g. 41000:1:0 in the following statements:

```
DEFINE_DEVICE
vdvModule = 41000:1:0
dvDevice =  135:1:0
DEFINE_MODULE 'LightModule' LightModule1 (vdvModule , dvDevice )
```

While it is possible to create a Duet Virtual Device on a port other than 1 and pass it to the Duet module, the behavior of the module is undefined.

Channel and Level Ranges

SNAPI uses only channels in range 1-299. Some channels are used for multiple functions but these channels belong to components that do not overlap within a single device. For instance, HVAC and Display both use channel 214 for setFanState and setFreezeOn respectively. Some channels are used for the same function in multiple components, for instance Video Conference and Display both define channel 191 for cyclePIPPosition. In both cases, this is by design.

Some devices may use custom channels for advanced functions. Channels 67-76 and 300-399 are reserved for modules to use for whatever functions they like. See specific module documentation for details on the channels used in that module.

SNAPI uses Levels in the range 1-48. Some levels are used for the same function in multiple components, for instance HVAC, Pool/Spa and Weather all define level 34 for Outdoor Temperature. This is by design.

Some device may use custom levels for advanced functions. Levels 50-80 and above are reserved for modules to use for whatever functions they like. See specific module documentation for details on the levels used in that module.

Commands and Escape Characters

SNAPI command uses comma as a parameter separator. If a parameter's value contains a comma, the parameter is escaping using double quotes at the start and end of the parameter. If a parameter's value contains a double quote character it is escaped with a pair of double quote characters.

The following examples are properly escaped parameter values:

- 6
- Hello
- Brown Eyed Girl
- "Morrison, Van"
- "Van ""The Man"" Morrison"

The following examples are improperly escaped parameter values:

- Morrison, Van
- Van "The Man" Morrison

SNAPI.axi includes a few helpful routines to build commands:

- DuetPackCmdHeader(Hdr)
- DuetPackCmdParam(Cmd, Param)
- DuetPackCmdParamArray(Cmd, Params[])

DuetPackCmdHeader is a command using a given command header where Hdr is the command header. DuetPackCmdParam adds a parameter to the command, escaping the parameter and adding parameter separators as needed; Cmd is the command to which the parameter is added and Param is the parameter to be added. DuetPackCmdParamArray is similar to DuetPackCmdParam but it takes an array of parameters and adds them to the command. All of these functions return the updated command.

SNAPI.axi includes a few helpful routines to parse commands as well:

- DuetParseCmdHeader(Cmd)
- DuetParseCmdParam(Cmd)

DuetParseCmdHeader removes and returns the command header from a command. DuetParseCmdParam removes and returns the next parameter from the command, un-escaping the parameter as needed. Both of these functions return a string containing the command header or the parameter.

An example program using these routines is shown below:

```
// Build a command to be stored in cTestCmd
cTestCmd = DuetPackCmdHeader('COMMAND')
cTestCmd = DuetPackCmdParam(cTestCmd,'Morrison,Van')
cTestCmd = DuetPackCmdParam(cTestCmd,'Wild Nights')
cTestCmd = DuetPackCmdParam(cTestCmd,'"The Man"')
cTestCmd = DuetPackCmdParam(cTestCmd,'Tupelo Honey')

// Resulting command is:
// 'COMMAND-"Morrison, Van",Wild Nights,""The Man"",Tupelo Honey'

// Remove the parameters for this command
cCmdheader = DuetParseCmdHeader(cTestCmd)
SWITCH (cCmdheader)
{
CASE 'COMMAND':
   {
   cParam1 = DuetParseCmdParam(cTestCmd)
   cParam2 = DuetParseCmdParam(cTestCmd)
   cParam3 = DuetParseCmdParam(cTestCmd)
   cParam4 = DuetParseCmdParam(cTestCmd)

   // cParam1 = 'Morrison, Van'
   // cParam2 = 'Wild Nights'
   // cParam3 = '"The Man"'
   // cParam4 = 'Tupelo Honey'
   }
}
```

Amplifier

Component					
Name: Amplifier					
Interface: IAmplifierComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Amplifier Listener					
Interface: IAmplifierComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Audio Conferencer

Component					
Name: Audio Conferencer					
Interface: IAudioConferencerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cyclePrivacy()`	145			`ACONF_PRIVACY`	Momentary Function Channel: Cycle privacy when channel is activated
`setPrivacyOn(state)`	146			`ACONF_PRIVACY_ON`	Discrete Function Channel: Privacy is on while channel is active
`train()`	147			`ACONF_TRAIN`	Momentary Function Channel: Train is executed when the channel is activated

Listener					
Name: Audio Conferencer Listener					
Interface: IAudioConferencerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processPrivacyEvent`	146			`ACONF_PRIVACY_FB`	Feedback Channel: Privacy is muted if channel is active.

Audio Mixer

Component					
Name: Audio Mixer					
Interface: IAudioMixerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getAudioMixerCrosspoint(input,output)`			`?XPOINT-<input>,<output>`		Query for Audio Mixer Crosspoint, responds with XPOINT-<value>,<input>,<output> where <value> is 0 to 255, <input> is 1 to the maximum supported input and <output> is the maximum supported output (see module documentation)
`getAudioMixerPreset()`			`?MIXERPRESET`		Query for Audio Mixer Preset, responds with MIXERPRESET-<preset> where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
`isAudioMixerCrosspointMuteOn(input,output)`			`?XPOINTMUTE-<input>,<output>`		Query for Audio Mixer Crosspoint Mute, responds with XPOINTMUTE-<state>,<input>,<output> where <state> is 0 (un-muted) or 1 (muted), <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`saveAudioMixerPreset(preset)`			`MIXERPRESETSAVE-<preset>`		Save Audio Mixer Preset where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
`setAudioMixerCrosspoint(input,output[],value)`			`XPOINT-<value>,<input>,<output,...>`		Set Audio Mixer Crosspoint for <input> to one or more <output>s where <value> is 0 to 255. <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`setAudioMixerCrosspointMuteOn(input,output,state)`			`XPOINTMUTE-<state>,<input>,<output>`		Set Audio Mixer Crosspoint Mute for <input> and <output> where <state> is 0 (un-muted) or 1 (muted). <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`setAudioMixerPreset(preset)`			`MIXERPRESET-<preset>`		Recall Audio Mixer Preset where <preset> is 1 to x and x is the maximum supported preset (see module documentation)

Listener					
Name: Audio Mixer Listener					
Interface: IAudioMixerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processAudioMixerCrosspointEvent`			`XPOINT-<value>,<input>,<output>`		Audio Mixer Crosspoint changed for <input> to one or more <output>s where <value> is 0 to 255. <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`processAudioMixerCrosspointMuteOnEvent`			`XPOINTMUTE-<state>,<input>,<output>`		Audio Mixer Crosspoint Mute changed for <input> and <output> where <state> is 0 (un-muted) or 1 (muted). <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`processAudioMixerPresetEvent`			`MIXERPRESET-<preset>`		Mixer preset changed, where <preset> is 1 to x and x is the maximum supported preset (see module documentation)

Audio Processor

Component					
Name: Audio Processor					
Interface: IAudioProcessorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
addAudioProcessorComponent (index,audioProcessorAddress)			AUDIOPROCADD-<index>, <audioProcessorAddress>		Add a Audio Processor object at a given index, where <index> is 1 through x and <address> is the object address and x is the maximum supported index (see module documentation)
adjustAudioProcessorLevel(1)	24			AUDIOPROC_LEVEL_UP	Ramping Channel: Audio Processor level is incremented when channel is activated
adjustAudioProcessorLevel(-1)	25			AUDIOPROC_LEVEL_DN	Ramping Channel: Audio Processor level is decremented when channel is activated
cycleAudioProcessorPreset()	209			AUDIOPROC_PRESET	Momentary Function Channel: Cycle Audio Processor preset when channel is activated
cycleAudioProcessorState()	26			AUDIOPROC_STATE	Momentary Function Channel: Cycle Audio Processor state when channel is activated
getAudioProcessorComponentAddress (index)			?AUDIOPROCADDR-<index>		Query for the address of the Audio Processor object at index <index>, responds with AUDIOPROCADDR-<index>,<address>
getAudioProcessorComponentIndex (audioProcessorAddress)			?AUDIOPROCIDX-<address>		Query for the index of the Audio Processor object with address <address>, responds with AUDIOPROCADDR-<index>, <address>
getAudioProcessorCrosspoint (input,output)			?XPOINT-<input>,<output>		Query for Audio Processor crosspoint, responds with XPOINT-<value>,<input>,<output> where <value> is 0 to 255, <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
getAudioProcessorInput(output)			?INPUT-<output>		Query for the input connected to an output, respond with SWITCH-L<sl>I<input>O<output> where <sl> is AUDIO and <input> is 0 if there is no connection.
getAudioProcessorOutput(input)			?OUTPUT-<input>		Query for the outputs connected to an input, respond with SWITCH-L<sl>I<input>O<output>,<output>, where <sl> is AUDIO and <input> is 0 if there is no connection.
getAudioProcessorPreset()			?AUDIOPROCPRESET		Query for Audio Processor Preset, responds with AUDIOPROCPRESET-<preset> where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
isAudioProcessorCrosspointMuteOn (input,output)			?XPOINTMUTE-<input>,<output>		Query for Audio Processor Crosspoint Mute, responds with XPOINTMUTE-<state>,<input>,<output> where <state> is 0 (un-muted) or 1 (muted), <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)

Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`removeAudioProcessorComponent (audioProcessorAddress)`			`AUDIOPROCREMOVEADDR-<audioProcessorAddress>`		Remove the Audio Processor object with address <address>, where <address> is the object address
`removeAudioProcessorComponent (index)`			`AUDIOPROCREMOVEIDX-<index>`		Remove the Audio Processor object at index <index>, where <index> is 1 through x and x is the maximum supported index (see module documentation)
`saveAudioProcessorPreset(preset)`			`AUDIOPROCPRESETSAVE-<preset>`		Save Audio Processor Preset where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
`setAudioProcessorCrosspoint (input,output[],value)`			`XPOINT-<value>,<input>,<output,...>`		Set Audio Processor crosspoint for <input> to one or more <output>s where <value> is 0 to 255. <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`setAudioProcessorCrosspointMuteOn (input,output,state)`			`XPOINTMUTE-<state>,<input>,<output>`		Set Audio Processor Crosspoint Mute for <input> and <output> where <state> is 0 (un-muted) or 1 (muted). <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`setAudioProcessorLevel(level)`		1		`AUDIOPROC_LVL`	Set Audio Processor level, range is 0-255
`setAudioProcessorPreset(preset)`			`AUDIOPROCPRESET-<preset>`		Recall Audio Processor Preset where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
`setAudioProcessorStateOn(state)`	199			`AUDIOPROC_STATE_ON`	Discrete Function Channel: Audio Processor state is on while channel is active
`switchAudioProcessorInputToOutput (input,output[])`			`AI<input>O<output,...>`		Switch <input> to one or more <output>s for switcher level Audio. Use <input> 0 for disconnect.

Listener					
Name: Audio Processor Listener					
Interface: IAudioProcessorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processAudioProcessorCrosspointEvent`			`XPOINT-<value>,<input>,<output>`		Audio Processor crosspoint changed for <input> to one or more <output>s where <value> is 0 to 255. <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`processAudioProcessorCrosspoint MuteOnEvent`			`XPOINTMUTE-<state>,<input>,<output>`		Audio Processor Crosspoint Mute changed for <input> and <output> where <state> is 0 (un-muted) or 1 (muted). <input> is 1 to the maximum supported input and <output> is 1 to the maximum supported output (see module documentation)
`processAudioProcessorLevelEvent`		1		`AUDIOPROC_LVL`	Audio Processor level changed, range is 0-255
`processAudioProcessorPresetEvent`			`AUDIOPROCPRESET-<preset>`		Audio Processor preset changed, where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
`processAudioProcessorStateOnEvent`	199			`AUDIOPROC_STATE_FB`	Feedback Channel: Audio Processor state is on if channel is on
`processAudioProcessorSwitchEvent`			`SWITCH-L<sl>I<input>O<output>`		Audio Processor switch connections changed, where <sl> is AUDIO and <input> is 0 if there is no connection.

Audio Tape

Component					
Name: Audio Tape					
Interface: IAudioTapeComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleAudioTapeRecordMute()`	128			`CASS_RECORD_MUTE`	Momentary Function Channel: Cycle record mute when channel is activated
`cycleAudioTapeSide()`	42			`CASS_TAPE_SIDE`	Momentary Function Channel: Cycle tape side when channel is activated
`reversePlay()`	41			`CASS_REVERSE_PLAY`	Momentary Function Channel: Reverse direction of play (but not the side). The audio will be played backwards.
`setAudioTapeRecordMuteOn(state)`	200			`CASS_RECORD_MUTE_ON`	Discrete Function Channel: Record Mute is on while channel is active
`setAudioTapeSide(A)`	126			`CASS_TAPE_SIDE_A`	Momentary Function Channel: Set tape side to side A
`setAudioTapeSide(B)`	127			`CASS_TAPE_SIDE_B`	Momentary Function Channel: Set tape side to side B

Listener					
Name: Audio Tape Listener					
Interface: IAudioTapeComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processAudioTapeRecordMuteOnEvent`	200			`CASS_RECORD_MUTE_FB`	Feedback Channel: Record mute is on while channel is active
`processAudioTapeSideEvent`	126			`CASS_TAPE_SIDE_A_FB`	Feedback Channel: Tape side is set to side A
`processAudioTapeSideEvent`	127			`CASS_TAPE_SIDE_B_FB`	Feedback Channel: Tape side is set to side B

Audio Tuner Device

Component					
Name: Audio Tuner Device					
Interface: IAudioTunerDeviceComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Audio Tuner Device Listener					
Interface: IAudioTunerDeviceComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Camera

Component					
Name: Camera					
Interface: ICameraComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleAutoFocus()`	172			`AUTO_FOCUS`	Momentary Function Channel: Cycle auto focus when channel is activated
`cycleAutoIris()`	173			`AUTO_IRIS`	Momentary Function Channel: Cycle auto iris when channel is activated
`cycleCameraPreset()`	177			`CAM_PRESET`	Momentary Function Channel: Cycle camera preset when channel is activated
`getCameraComponentCount()`			`?CAMERACOMPONENTCOUNT`		Query number of camera components, responds with CAMERACOMPONENTCOUNT-<count>
`getCameraPreset()`			`?CAMERAPRESET`		Query for camera preset, responds with CAMERAPRESET-<preset>
`getCameraPresetCount()`			`?CAMERAPRESETCOUNT`		Query number of presets on a camera, responds with CAMERAPRESETCOUNT-<count>
`getCameraPresetProperties()`			`?CAMERAPRESETPROPERTIES`		Query properties for every camera preset, responds with CAMERAPRESETPROPERTIES-<index>,<displayName> [;<index>,<displayName>]
`getCameraPresetProperty(index)`			`?CAMERAPRESETPROPERTY`		Query properties for a single camera preset, responds with CAMERAPRESETPROPERTY-<index>,<displayName>
`saveCameraPreset(preset)`			`CAMERAPRESETSAVE-<preset>`		Save Camera Preset where <preset> is 1 to x and x is the maximum supported preset (see specific module documentation)
`setAutoFocusOn(state)`	162			`AUTO_FOCUS_ON`	Discrete Function Channel: Auto focus is on while channel is active
`setAutoIrisOn(state)`	163			`AUTO_IRIS_ON`	Discrete Function Channel: Auto iris is on while channel is active
`setCameraPreset(preset)`			`CAMERAPRESET-<preset>`		Recall camera preset where <preset> is 1-x and x is the maximum supported preset (see specific module documentation)
`setFocus(focus)`		16		`FOCUS_LVL`	Set focus position, range is 0-255 (0=near)
`setFocusRamp(FAR)`	161			`FOCUS_FAR`	Ramping Channel: Focus is ramped far while channel is active
`setFocusRamp(NEAR)`	160			`FOCUS_NEAR`	Ramping Channel: Focus is ramped near while channel is active
`setFocusSpeed(speed)`		19		`FOCUS_SPEED_LVL`	Set focus speed, range is 0-255 (0=slow)
`setIris(iris)`		17		`IRIS_LVL`	Set iris position, range is 0-255 (0=closed)
`setIrisRamp(CLOSE)`	175			`IRIS_CLOSE`	Ramping Channel: Iris is ramped closed while channel is active
`setIrisRamp(OPEN)`	174			`IRIS_OPEN`	Ramping Channel: Iris is ramped open while channel is active
`setIrisSpeed(speed)`		20		`IRIS_SPEED_LVL`	Set iris speed, range is 0-255 (0=slow)
`setPan(pan)`		27		`PAN_LVL`	Set pan position, range is 0-255 (0=left)
`setPanRamp(LEFT)`	134			`PAN_LT`	Ramping Channel: Pan is ramped left while channel is active

Component Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
`setPanRamp(RIGHT)`	135			`PAN_RT`	Ramping Channel: Pan is ramped right while channel is active
`setPanSpeed(speed)`		29		`PAN_SPEED_LVL`	Set pan speed, range is 0-255 (0=slow)
`setTilt(tilt)`		28		`TILT_LVL`	Set tilt position, range is 0-255 (0=down)
`setTiltRamp(DOWN)`	133			`TILT_DN`	Ramping Channel: Tilt is ramped down while channel is active
`setTiltRamp(UP)`	132			`TILT_UP`	Ramping Channel: Tilt is ramped up while channel is active
`setTiltSpeed(speed)`		30		`TILT_SPEED_LVL`	Set tilt speed, range is 0-255 (0=slow)
`setZoom(zoom)`		15		`ZOOM_LVL`	Set zoom position, range is 0-255 (0=out/Wide)
`setZoomRamp(IN)`	159			`ZOOM_IN`	Ramping Channel: Zoom is ramped in (tele) while channel is active
`setZoomRamp(OUT)`	158			`ZOOM_OUT`	Ramping Channel: Zoom is ramped out (wide) while channel is active
`setZoomSpeed(speed)`		18		`ZOOM_SPEED_LVL`	Set zoom speed, range is 0-255 (0=slow)

Listener					
Name: Camera Listener					
Interface: ICameraComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processAutoFocusEvent	162			AUTO_FOCUS_FB	Feedback Channel: Auto focus is on if channel is active
processAutoIrisEvent	163			AUTO_IRIS_FB	Feedback Channel: Auto iris is on if channel is active
processCameraComponentCountEvent			CAMERACOMPONENTCOUNT-<count>		Returns integer number of camera components
processCameraPresetCountEvent			CAMERAPRESETCOUNT-<count>		Returns integer number of camera presets
processCameraPresetEvent			CAMERAPRESET-<preset>		Camera preset changed, where <preset> is 1-x and x is the maximum supported preset (see module documentation)
processCameraPresetPropertiesEvent			CAMERAPRESETPROPERTIES-<index>,<displayName>[;-<index>,<displayName>,]*		Returns string containing <index> and <displayName> for each preset.
processCameraPresetPropertyEvent			CAMERAPRESETPROPERTY-<index>,<displayName>		Returns string containing <index> and <displayName> for a preset.
processFocusEvent		16		FOCUS_LVL	Focus changed, range is 0-255 (0=near)
processFocusRampEvent	161			FOCUS_FAR_FB	Feedback Channel: Focus is ramping far while channel is on
processFocusRampEvent	160			FOCUS_NEAR_FB	Feedback Channel: Focus is ramping near while channel is on
processFocusSpeedEvent		19		FOCUS_SPEED_LVL	Focus speed changed, range is 0-255 (0=slow)
processIrisEvent		17		IRIS_LVL	Iris changed, range is 0-255 (0=closed)
processIrisRampEvent	175			IRIS_CLOSE_FB	Feedback Channel: Iris is ramping closed while channel is on
processIrisRampEvent	174			IRIS_OPEN_FB	Feedback Channel: Iris is ramping open while channel is on
processIrisSpeedEvent		20		IRIS_SPEED_LVL	Iris speed changed, range is 0-255 (0=slow)
processPanEvent		27		PAN_LVL	Pan changed, range is 0-255 (0=left)
processPanRampEvent	134			PAN_LT_FB	Feedback Channel: Pan is ramping left while channel is on
processPanRampEvent	135			PAN_RT_FB	Feedback Channel: Pan is ramping right up while channel is on
processPanSpeedEvent		29		PAN_SPEED_LVL	Pan speed changed, range is 0-255 (0=slow)
processTiltEvent		28		TILT_LVL	Tilt changed, range is 0-255 (0=down)
processTiltRampEvent	133			TILT_DN_FB	Feedback Channel: Tilt is ramping down while channel is on
processTiltRampEvent	132			TILT_UP_FB	Feedback Channel: Tilt is ramping up while channel is on
processTiltSpeedEvent		30		TILT_SPEED_LVL	Tilt speed changed, range is 0-255 (0=slow)
processZoomEvent		15		ZOOM_LVL	Zoom changed, range is 0-255 (0=out/wide)
processZoomRampEvent	159			ZOOM_IN_FB	Feedback Channel: Zoom is ramping in (tele) while channel is on
processZoomRampEvent	158			ZOOM_OUT_FB	Feedback Channel: Zoom is ramping out (wide) while channel is on
processZoomSpeedEvent		18		ZOOM_SPEED_LVL	Zoom speed changed, range is 0-255 (0=slow)

Dialer

Component					
Name: Dialer					
Interface: IDialerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleAudibleRing()`	205			`DIAL_AUDIBLE_RING`	Momentary Function Channel: Cycle audible ring when channel is activated
`cycleAutoAnswer()`	204			`DIAL_AUTO_ANSWER`	Momentary Function Channel: Cycle auto answer when channel is activated
`cycleOffHook()`	202			`DIAL_OFF_HOOK`	Momentary Function Channel: Cycle hook state when channel is activated
`dial(index)`			`DIALINDEX-<index>`		Dial a speed dial index, where <index> is 1 to x and x is the maximum supported speed dial index (see module documentation)
`dial(recordID)`			`DIALID-<recordID>`		Dial a speed dial record.
`dialDTMF(char)`			`DTMF-<digit>`		Send a DTMF tone for a character without regard for hook status
`dialNumber(number)`			`DIALNUMBER-<number>`		Dial a number where <number> is the number to be dialed.
`flashHook()`	208			`DIAL_FLASH_HOOK`	Momentary Function Channel: Flash hook when channel is activated
`getDialerStatus()`			`?DIALERSTATUS`		Query dialer status, responds with DIALERSTATUS-<status>, where <status> is DIALING, BUSY, RINGING, DISCONNECTED, NEGOTIATING, FAULT, CONNECTED
`redial()`	201			`DIAL_REDIAL`	Momentary Function Channel: Redial when channel is activated
`setAudibleRingOn(state)`	240			`DIAL_AUDIBLE_RING_ON`	Discrete Function Channel: Audible ring is on while channel is active
`setAutoAnswerOn(state)`	239			`DIAL_AUTO_ANSWER_ON`	Discrete Function Channel: Auto answer is on while channel is active
`setOffHook(state)`	238			`DIAL_OFF_HOOK_ON`	Discrete Function Channel: Hook state is off hook while channel is active

Listener					
Name: Dialer Listener					
Interface: IDialerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processAudibleRingOnEvent	240			DIAL_AUDIBLE_RING_FB	Feedback Channel: Audible ring is on if channel is on
processAutoAnswerOnEvent	239			DIAL_AUTO_ANSWER_FB	Feedback Channel: Auto answer is on if channel is on
processDialerStatusEvent			DIALERSTATUS-BUSY		Dialer status changed, number being dialed is busy
processDialerStatusEvent			DIALERSTATUS-CONNECTED		Dialer status changed, dialer is connected
processDialerStatusEvent			DIALERSTATUS-DIALING		Dialer status changed, dialer is dialing
processDialerStatusEvent			DIALERSTATUS-DISCONNECTED		Dialer status changed, dialer is disconnected/idle
processDialerStatusEvent			DIALERSTATUS-FAULT		Dialer status changed, dialer encounter a fault during dialing/negotiating
processDialerStatusEvent			DIALERSTATUS-NEGOTIATING		Dialer status changed, dialer is negotiating
processDialerStatusEvent			DIALERSTATUS-RINGING		Dialer status changed, number being dialed is ringing
processIncomingCallEvent			INCOMINGCALL-<number>		An incoming call is detected. If available via caller ID, the phone number will be supplied
processOffHookEvent	238			DIAL_OFF_HOOK_FB	Feedback Channel: Hook state is off hook if channel is on

Digital Media Decoder

Component					
Name: Digital Media Decoder					
Interface: IDigitalMediaDecoderComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Digital Media Decoder Listener					
Interface: IDigitalMediaDecoderComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Digital Media Encoder

Component					
Name: Digital Media Encoder					
Interface: IDigitalMediaEncoderComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Digital Media Encoder Listener					
Interface: IDigitalMediaEncoderComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Digital Media Server

Component					
Name: Digital Media Server					
Interface: IDigitalMediaServerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Digital Media Server Listener					
Interface: IDigitalMediaServerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Digital Satellite System

Component					
Name: Digital Satellite System					
Interface: IDigitalSatelliteSystemComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Digital Satellite System Listener					
Interface: IDigitalSatelliteSystemComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Digital Video Recorder

Component					
Name: Digital Video Recorder					
Interface: IDigitalVideoRecorderComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Digital Video Recorder Listener					
Interface: IDigitalVideoRecorderComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Disc Device

Component					
Name: Disc Device					
Interface: IDiscDeviceComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleDiscTray()`	120			`DISC_TRAY`	Momentary Function Channel: Open or Close the disc tray when the channel is activated
`cycleRandom()`	124			`DISC_RANDOM`	Momentary Function Channel: Cycle random when channel is activated
`cycleRepeat()`	125			`DISC_REPEAT`	Momentary Function Channel: Cycle repeat when channel is activated
`getDiscCapacity()`			`?DISCCAPACITY`		Query for the disc capacity responds with DISCCAPACITY-<discs> when <discs> is the number of disc slots supported by the device (see module documentation)
`getDiscInfo()`			`?DISCINFO`		Query for disc info, responds with DISCINFO-<num>,<duration>,<# of
`getTitleInfo()`			`?TITLEINFO`		Query for title info changed, responds with TITLEINFO- <num>,<duration>,<# of
`nextDisc()`	55			`DISC_NEXT`	Momentary Function Channel: Set disc next when channel is activated
`previousDisc()`	56			`DISC_PREV`	Momentary Function Channel: Set disc previous when channel is activated
`queryDiscProperties()`			`?DISCPROPS`		Query for the disc properties, responds with multiple DISCPROP-<key>,<value> commands, one for each key
`queryDiscProperty(key)`			`?DISCPROP-<key>`		Query for a disc property, responds with DISCPROP-<key>,<value> command
`queryTitleProperties()`			`?TITLEPROPS`		Query for the title properties, responds with multiple TITLEPROP-<key>,<value> commands, one for each key
`queryTitleProperty(key)`			`?TITLEPROP-<key>`		Query for a title property, responds with TITLEPROP-<key>,<value> command
`setDisc(discNumber)`			`SETDISC-<discNumber>`		Set disc number to <discNumber>, where <discNumber> is 1 to <x> where x is the disc capacity (see getDiscCapactiy() or module documentation)
`setPlayPosition(titleNumber, trackNumber)`			`PLAYPOSITION-<titleNumber>, <trackNumber>`		Set the play position where <titleNumber> is the Title number and <trackNumber> is the Track number
`setPlayPosition(titleNumber, trackNumber,relativeCounter)`			`PLAYPOSITION-<titleNumber>, <trackNumber>,<relativeCounter>`		Set the play position where <titleNumber> is the Title number, <trackNumber> is the Track number and <relativeCounter> is a String in the format [-]hh:mm:ss.ff, mm should be 0 >= mm < 60, ss should be 0 >= ss < 60, ff should be valid for the disc type

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`setPlayPosition(trackNumber)`			`PLAYPOSITION-<trackNumber>`		Set the play position where <trackNumber> is the Track number
`setPlayPosition(trackNumber, relativeCounter)`			`PLAYPOSITION-<trackNumber>, <relativeCounter>`		Set the play position where <track> is the Track number and <counter> is a String in the format [-]hh:mm:ss.ff, mm should be 0 >= mm < 60, ss should be 0 >= ss < 60, ff should be valid for the disc type
`setRandomState(RANDOM_ALL)`	179			`DISC_RANDOM_ALL_ON`	Momentary Function Channel: Random-all is on while channel is active
`setRandomState(RANDOM_DISC)`	178			`DISC_RANDOM_DISC_ON`	Momentary Function Channel: Random-disc is on while channel is active
`setRandomState(RANDOM_OFF)`	180			`DISC_RANDOM_OFF_ON`	Momentary Function Channel: Random-off is on while channel is active
`setRepeatState(REPEAT_ALL)`	183			`DISC_REPEAT_ALL_ON`	Momentary Function Channel: Repeat-all is on while channel is active
`setRepeatState(REPEAT_DISC)`	181			`DISC_REPEAT_DISC_ON`	Momentary Function Channel: Repeat-disc is on while channel is active
`setRepeatState(REPEAT_OFF)`	184			`DISC_REPEAT_OFF_ON`	Momentary Function Channel: Repeat-off is on while channel is active
`setRepeatState(REPEAT_TRACK)`	182			`DISC_REPEAT_TRACK_ON`	Momentary Function Channel: Repeat-track is on while channel is active
`setTitleCounterNotificationOn (state)`			`TITLECOUNTERNOTIFY-<state>`		Turn title counter notification on or off, where <state> is 1 or 0

Listener					
Name: Disc Device Listener					
Interface: IDiscDeviceComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processDiscInfoEvent`			`DISCINFO-<num>,<duration>, <totalTitles>,<totalTracks>, <discType>,<id>`		Disc info changed, where <num> is disc number, <duration> is a disc duration String in the format [-]hh:mm:ss.ff, <# of tracks> is the number of titles in this disc, <# of tracks> is the number of tracks in this disc, <disctype> is the disc type (AUDIO_ONLY, VIDEO_ONLY, AUDIOVIDEO) and <id> is the disc database id.
`processDiscPropertiesEvent`			`DISCPROP-<key>,<value>`		Disc properties query changed, responds with multiple DISCPROP-<key>,<value> commands, one for each key
`processRandomStateEvent`	179			`DISC_RANDOM_ALL_FB`	Feedback Channel: Random state change (see chart below)
`processRandomStateEvent`	178			`DISC_RANDOM_DISC_FB`	Feedback Channel: Random state change (see chart below)
`processRandomStateEvent`	180			`DISC_RANDOM_OFF_FB`	Feedback Channel: Random state change (see chart below)
`processRepeatStateEvent`	183			`DISC_REPEAT_ALL_FB`	Feedback Channel: Repeat state change (see chart below)
`processRepeatStateEvent`	181			`DISC_REPEAT_DISC_FB`	Feedback Channel: Repeat state change (see chart below)
`processRepeatStateEvent`	184			`DISC_REPEAT_OFF_FB`	Feedback Channel: Repeat state change (see chart below)
`processRepeatStateEvent`	182			`DISC_REPEAT_TRACK_FB`	Feedback Channel: Repeat state change (see chart below)
`processTitleCounterEvent`			`TITLECOUNTER-<counter>`		Title counter changed, where <counter> is a String in the format [-]hh:mm:ss.ff

Listener Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processTitleInfoEvent			TITLEINFO-<num>,<duration>, <# of tracks>,<discNumber>		Title info changed, where <num> is title number, <duration> is a title duration String in the format [-]hh:mm:ss.ff, <# of tracks> is the number of tracks in this title and <discNumber> is the disc number the title belongs to.
processTitlePropertiesEvent			TITLEPROP-<key>,<value>		Title properties query response, responds with multiple TITLEPROP-<key>,<value> commands, one for each key

Disc Device State Charts

processRandomStateEvent			
State	**Channel 178**	**Channel 179**	**Channel 180**
RANDOM_DISC	ON	OFF	OFF
RANDOM_ALL	OFF	ON	OFF
RANDOM_OFF	OFF	OFF	ON

processRepeatStateEvent				
State	**Channel 181**	**Channel 182**	**Channel 183**	**Channel 184**
REPEAT_DISC	ON	OFF	OFF	OFF
REPEAT_TRACK	OFF	ON	OFF	OFF
REPEAT_ALL	OFF	OFF	ON	OFF
REPEAT_OFF	OFF	OFF	OFF	ON

processDiscTransportEvent								
State	**Channel 241**	**Channel 242**	**Channel 243**	**Channel 246**	**Channel 247**	**Channel 248**	**Channel 249**	**Channel 250**
PLAY	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
STOP	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
PAUSE	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
SCAN_FWD	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
SCAN_REV	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
RECORD	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF
RECORD_PAUSE	OFF	OFF	ON	OFF	OFF	ON	OFF	OFF
SLOW_FWD	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
SLOW_REV	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON

processPowerEvent	
State	**Channel 255**
OFF	OFF
ON	ON

Disc Device Listener State Charts

Listener States			
State	**Channel 178**	**Channel 179**	**Channel 180**
RANDOM_DISC	ON	OFF	OFF
RANDOM_ALL	OFF	ON	OFF
RANDOM_OFF	OFF	OFF	ON

processRepeatStateEvent				
State	**Channel 181**	**Channel 182**	**Channel 183**	**Channel 184**
REPEAT_DISC	ON	OFF	OFF	OFF
REPEAT_TRACK	OFF	ON	OFF	OFF
REPEAT_ALL	OFF	OFF	ON	OFF
REPEAT_OFF	OFF	OFF	OFF	ON

Disc Transport

Component					
Name: Disc Transport					
Interface: IDiscTransportComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleScanSpeed()`	192			`SCAN_SPEED`	Momentary Function Channel: Cycles the scan speed when the channel is activated
`getTrackInfo()`			`?TRACKINFO`		Query for the track info, responds with TRACKINFO-<num>,<duration>,<discNumber>, where <duration> is a String in the format hh:mm:ss.ff
`queryTrackProperties()`			`?TRACKPROPS`		Query for the track properties, responds with multiple TRACKPROP-<key>,<value> commands, one for each key
`queryTrackProperty(key)`			`?TRACKPROP-<keyName>`		Query for a track property, responds with TRACKPROP-<key>,<value> command
`setDiscTransport(FRAME_FWD)`	185			`FRAME_FWD`	Momentary Function Channel: Deck is set to step frame forward when the channel is activated
`setDiscTransport(FRAME_REV)`	186			`FRAME_REV`	Momentary Function Channel: Deck is set to step frame reverse when the channel is activated
`setDiscTransport(NEXT)`	4			`FFWD`	Momentary Function Channel: Deck is set to next track/chapter when the channel is activated
`setDiscTransport(PAUSE)`	3			`PAUSE`	Momentary Function Channel: Deck is set to pause when the channel is activated
`setDiscTransport(PLAY)`	1			`PLAY`	Momentary Function Channel: Deck is set to play when the channel is activated
`setDiscTransport(PREVIOUS)`	5			`REW`	Momentary Function Channel: Deck is set to previous track/chapter when the channel is activated
`setDiscTransport(RECORD)`	8			`RECORD`	Momentary Function Channel: Deck is set to record when the channel is activated
`setDiscTransport(SCAN_FWD)`	6			`SFWD`	Momentary Function Channel: Deck is set to scan forward when the channel is activated
`setDiscTransport(SCAN_REV)`	7			`SREV`	Momentary Function Channel: Deck is set to scan reverse when the channel is activated
`setDiscTransport(SLOW_FWD)`	188			`SLOW_FWD`	Momentary Function Channel: Deck is set to slow forward when the channel is activated
`setDiscTransport(SLOW_REV)`	189			`SLOW_REV`	Momentary Function Channel: Deck is set to slow reverse when the channel is activated
`setDiscTransport(STOP)`	2			`STOP`	Momentary Function Channel: Deck is set to stop when the channel is activated
`setPlayPosition(mt)`			`PLAYPOSITION-<counter>`		Set the play position where <counter> is a String in the format [-]hh:mm:ss.ff, mm should be 0 >= mm < 60, ss should be 0 >= ss < 60, ff should be valid for the disc type
`setTrackCounterNotificationOn (state)`			`TRACKCOUNTERNOTIFY-<state>`		Turn track counter notification on or off, where <state> is 1 or 0

Listener					
Name: Disc Transport Listener					
Interface: IDiscTransportComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processDiscTransportEvent`	243			`PAUSE_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	241			`PLAY_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	248			`RECORD_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	246			`SFWD_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	247			`SREV_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	249			`SLOW_FWD_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	250			`SLOW_REV_FB`	Feedback Channel: Transport state change (see chart below)
`processDiscTransportEvent`	242			`STOP_FB`	Feedback Channel: Transport state change (see chart below)
`processTrackCounterEvent`			`TRACKCOUNTER-<counter>`		Track counter changed, where <counter> is a String in the format [-]hh:mm:ss.ff
`processTrackInfoEvent`			`TRACKINFO-<num>, <duration>,<discNumber>`		Track info changed, where <num> is track number, <duration> is a track duration String in the format [-]hh:mm:ss.ff and <discNumber> is the disc number the track belongs to.
`processTrackPropertiesEvent`			`TRACKPROP-<key>,<value>`		Track properties query response, responds with multiple TRACKPROP-<key>, <value> commands, one for each key

Disc Transport Listener State Charts

processDiscTransportEvent								
State	**Channel 241**	**Channel 242**	**Channel 243**	**Channel 246**	**Channel 247**	**Channel 248**	**Channel 249**	**Channel 250**
PLAY	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
STOP	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
PAUSE	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
SCAN_FWD	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
SCAN_REV	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
RECORD	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF
RECORD_PAUSE	OFF	OFF	ON	OFF	OFF	ON	OFF	OFF
SLOW_FWD	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
SLOW_REV	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON

Display

Component					
Name: Display					
Interface: IDisplayComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`adjustBrightness(1)`	148			`BRIGHT_UP`	Momentary Function Channel: Brightness is incremented when channel is activated
`adjustBrightness(-1)`	149			`BRIGHT_DN`	Momentary Function Channel: Brightness is decremented when channel is activated
`adjustColor(1)`	150			`COLOR_UP`	Momentary Function Channel: Color is incremented when channel is activated
`adjustColor(-1)`	151			`COLOR_DN`	Momentary Function Channel: Color is decremented when channel is activated
`adjustContrast(1)`	152			`CONTRAST_UP`	Momentary Function Channel: Contrast is incremented when channel is activated
`adjustContrast(-1)`	153			`CONTRAST_DN`	Momentary Function Channel: Contrast is decremented when channel is activated
`adjustSharpness(1)`	154			`SHARP_UP`	Momentary Function Channel: Sharpness is incremented when channel is activated
`adjustSharpness(-1)`	155			`SHARP_DN`	Momentary Function Channel: Sharpness is decremented when channel is activated
`adjustTint(1)`	156			`TINT_UP`	Momentary Function Channel: Tint is incremented when channel is activated
`adjustTint(-1)`	157			`TINT_DN`	Momentary Function Channel: Tint is decremented when channel is activated
`cycleAspectRatio()`	142			`ASPECT_RATIO`	Momentary Function Channel: Cycle aspect ratios when channel is activated
`cycleFreeze()`	213			`PIC_FREEZE`	Momentary Function Channel: Cycle freeze when channel is activated
`cyclePictureMute()`	210			`PIC_MUTE`	Momentary Function Channel: Cycle picture mute when channel is activated
`cyclePIP()`	194			`PIP`	Momentary Function Channel: Cycle PIP when channel is activated
`cyclePIPPosition()`	191			`PIP_POS`	Momentary Function Channel: Cycle PIP positions when channel is activated
`getActiveWindow()`			`?ACTIVEWINDOW`		Query active window, responds with ACTIVEWINDOW-<window>, where <window> is LEFT,RIGHT,MAIN,SUB

Component Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
getAspectRatio()			?ASPECT		Query aspect ratio, responds with ASPECT-<ratio>, where <ratio> is ANAMORPHIC, WIDESCREEN, NORMAL
getAspectRatioCount()			?ASPECTRATIOCOUNT		Query aspect ratio count, responds with ASPECTRATIOCOUNT-<count>
getAspectRatioProperties(index)			?ASPECTRATIOPROPERTIES		Query properties for all aspect ratios, responds with ASPECTRATIOPROPERTIES-<index>,<displayName>,<value>[;<index>,<displayName>,<value>]*
getAspectRatioProperty(index)			?ASPECTRATIOPROPERTY-<index>		Query properties for single aspect ratio, responds with ASPECTRATIOPROPERTY-<index>,<displayName>, <value>
getAspectRatioSelect()			?ASPECTRATIOSELECT		Gets the index of the currently selected aspect ratio property.
getVideoType()			?VIDEOTYPE		Query video type, responds with VIDEOTYPE-<type> where <type> is AUTO,NTSC,PAL,SECAM
setActiveWindow(mss)			ACTIVEWINDOW-<mss>		Set active window, where <mss> is LEFT, RIGHT, MAIN, SUB
setAspectRatio(aspectRatio)			ASPECT-<aspectRatio>		Set aspect ratio, where <aspectRatio> is ANAMORPHIC, WIDESCREEN, NORMAL
setAspectRatioSelect(index)			ASPECTRATIOSELECT-<index>		Sets the current aspect ratio, where <index> is an integer number between 1 and the value returned by ?ASPECTRATIOSELECT, responds with ASPECTRATIOSELECT-<index>
setBrightness(level)		10		BRIGHT_LVL	Set brightness level, range is 0-255
setColor(level)		11		COLOR_LVL	Set color level, range is 0-255
setContrast(level)		12		CONTRAST_LVL	Set contrast level, range is 0-255
setFreezeOn(state)	214			PIC_FREEZE_ON	Discrete Function Channel: Freeze is on while channel is active
setPictureMuteOn(state)	211			PIC_MUTE_ON	Discrete Function Channel: Picture Mute is on while channel is active
setPIPOn(state)	195			PIP_ON	Discrete Function Channel: PIP is on while channel is active
setSharpness(level)		13		SHARP_LVL	Set sharpness level, range is 0-255
setTint(level)		14		TINT_LVL	Set tint level, range is 0-255
setVideoType(vt)			VIDEOTYPE-<vt>		Set video type, where <vt> is AUTO,NTSC,PAL,SECAM
swapPIP()	193			PIP_SWAP	Momentary Function Channel: Swap PIP when channel is activated

Listener					
Name: Display Listener					
Interface: IDisplayComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processActiveWindowEvent			ACTIVEWINDOW-<window>		Active window changed, where <window> is LEFT,RIGHT,MAIN,SUB
processAspectRatioCountEvent			ASPECTRATIOCOUNT-<count>		Responds with aspect ratio count, where <count> is an integer value
processAspectRatioEvent			ASPECT-<ratio>		Aspect ratio changed, where <ratio> is ANAMORPHIC,WIDESCREEN,NORMAL
processAspectRatioPropertiesEvent			ASPECTRATIOPROPERTIES-<index>,<displayName>,<value>[;<index>,<displayName>,<value>,...]		Returns properties for each supported aspect ratio, where <index> is an integer, <displayName> is the device text and <value> is NORMAL, WIDESCREEN, ANAMORPHIC or unique to that device
processAspectRatioPropertyEvent			ASPECTRATIOPROPERTY-<index>,<displayName>,<value>		Returns properties for single aspect ratio, where <index> is an integer,<displayName> is the device text and <value> is NORMAL, WIDESCREEN, ANAMORPHIC or unique to that device
processAspectRatioSelectEvent			ASPECTRATIOSELECT-<index>		Returns <index> of currently selected aspect ratio.
processBrightnessEvent		10		BRIGHT_LVL	Brightness changed, range is 0-255
processColorEvent		11		COLOR_LVL	Color changed, range is 0-255
processContrastEvent		12		CONTRAST_LVL	Contrast changed, range is 0-255
processFreezeEvent	214			PIC_FREEZE_FB	Feedback Channel: Freeze is on if channel is on
processPictureMuteEvent	211			PIC_MUTE_FB	Feedback Channel: Picture is muted if channel is on
processPIPEvent	195			PIP_FB	Feedback Channel: PIP is on if channel is on
processSharpnessEvent		13		SHARP_LVL	Sharpness changed, range is 0-255
processTintEvent		14		TINT_LVL	Tint changed, range is 0-255
processVideoTypeEvent			VIDEOTYPE-<type>		Video type changed, where <type> is AUTO, NTSC, PAL, SECAM

Document Camera

Component					
Name: Document Camera					
Interface: IDocumentCameraComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleAutoFocus()`	172			`AUTO_FOCUS`	Momentary Function Channel: Cycle auto focus when channel is activated
`cycleAutoIris()`	173			`AUTO_IRIS`	Momentary Function Channel: Cycle auto iris when channel is activated
`cycleLight()`	176			`DOCCAM_LIGHT`	Momentary Function Channel: Cycle lights when channel is activated
`setAutoFocusOn(state)`	162			`AUTO_FOCUS_ON`	Discrete Function Channel: Auto focus is on while channel is active
`setAutoIrisOn(state)`	163			`AUTO_IRIS_ON`	Discrete Function Channel: Auto iris is on while channel is active
`setFocus(focus)`		16		`FOCUS_LVL`	Set focus position, range is 0-255 (0=near)
`setFocusRamp(FAR)`	161			`FOCUS_FAR`	Ramping Channel: Focus is ramped far while channel is active
`setFocusRamp(NEAR)`	160			`FOCUS_NEAR`	Ramping Channel: Focus is ramped near while channel is active
`setFocusSpeed(speed)`		19		`FOCUS_SPEED_LVL`	Set focus speed, range is 0-255 (0=slow)
`setIris(iris)`		17		`IRIS_LVL`	Set iris position, range is 0-255 (0=closed)
`setIrisRamp(CLOSE)`	175			`IRIS_CLOSE`	Ramping Channel: Iris is ramped closed while channel is active
`setIrisRamp(OPEN)`	174			`IRIS_OPEN`	Ramping Channel: Iris is ramped open while channel is active
`setIrisSpeed(speed)`		20		`IRIS_SPEED_LVL`	Set iris speed, range is 0-255 (0=slow)
`setLowerLightOn(state)`	197			`DOCCAM_LOWER_LIGHT_ON`	Discrete Function Channel: Lower light is on while channel is active
`setUpperLightOn(state)`	198			`DOCCAM_UPPER_LIGHT_ON`	Discrete Function Channel: Upper light is on while channel is active
`setZoom(zoom)`		15		`ZOOM_LVL`	Set zoom position, range is 0-255 (0=out/wide)
`setZoomRamp(IN)`	159			`ZOOM_IN`	Ramping Channel: Zoom is ramped in (tele) while channel is active
`setZoomRamp(OUT)`	158			`ZOOM_OUT`	Ramping Channel: Zoom is ramped out (far) while channel is active
`setZoomSpeed(speed)`		18		`ZOOM_SPEED_LVL`	Set zoom speed, range is 0-255 (0=slow)

Listener					
Name: Document Camera Listener					
Interface: IDocumentCameraComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processAutoFocusEvent	162			AUTO_FOCUS_FB	Feedback Channel: Auto focus is on if channel is active
processAutoIrisEvent	163			AUTO_IRIS_FB	Feedback Channel: Auto iris is on if channel is active
processFocusEvent		16		FOCUS_LVL	Focus changed, range is 0-255 (0=near)
processFocusRampEvent	161			FOCUS_FAR_FB	Feedback Channel: Focus is ramping far while channel is on
processFocusRampEvent	160			FOCUS_NEAR_FB	Feedback Channel: Focus is ramping near while channel is on
processFocusSpeedEvent		19		FOCUS_SPEED_LVL	Focus speed changed, range is 0-255 (0=slow)
processIrisEvent		17		IRIS_LVL	Iris changed, range is 0-255 (0=closed)
processIrisRampEvent	175			IRIS_CLOSE_FB	Feedback Channel: Iris is ramping closed while channel is on
processIrisRampEvent	174			IRIS_OPEN_FB	Feedback Channel: Iris is ramping open while channel is on
processIrisSpeedEvent		20		IRIS_SPEED_LVL	Iris speed changed, range is 0-255 (0=slow)
processLowerLightEvent	197			DOCCAM_LOWER_LIGHT_FB	Feedback Channel: Lower light is on if channel is active
processUpperLightEvent	198			DOCCAM_UPPER_LIGHT_FB	Feedback Channel: Upper light is on if channel is active
processZoomEvent		15		ZOOM_LVL	Zoom changed, range is 0-255 (0=out/wide)
processZoomRampEvent	159			ZOOM_IN_FB	Feedback Channel: Zoom is ramping in (tele) while channel is on
processZoomRampEvent	158			ZOOM_OUT_FB	Feedback Channel: Zoom is ramping out (wide) while channel is on
processZoomSpeedEvent		18		ZOOM_SPEED_LVL	Zoom speed changed, range is 0-255 (0=slow)

Gain

Component					
Name: Gain					
Interface: IGainComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleGainMute()`	144			`GAIN_MUTE`	Momentary Function Channel: Cycle gain mute when channel is activated
`setGain(level)`		5		`GAIN_LVL`	Set gain, range is 0-255
`setGainMuteOn(state)`	143			`GAIN_MUTE_ON`	Discrete Function Channel: Gain mute is on while channel is active
`setGainRamp(DOWN)`	141			`GAIN_DN`	Ramping Channel: Gain is ramped down while channel is active
`setGainRamp(UP)`	140			`GAIN_UP`	Ramping Channel: Gain is ramped up while channel is active

Listener					
Name: Gain Listener					
Interface: IGainComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processGainEvent`		5		`GAIN_LVL`	Gain changed, range is 0-255
`processGainMuteEvent`	143			`GAIN_MUTE_FB`	Feedback Channel: Gain is muted if channel is on
`processGainRampEvent`	141			`GAIN_DN_FB`	Feedback Channel: Gain is ramping down while channel is on
`processGainRampEvent`	140			`GAIN_UP_FB`	Feedback Channel: Gain is ramping up while channel is on

HVAC

Component					
Name: HVAC					
Interface: IHVACComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
addHVACComponent (index,hvacAddress)			HVACADD-<index>, <hvacAddress>		Add a thermostat at a given index, where <index> is 1-x and <address> is a thermostat address and x is the maximum supported thermostat index (see module documentation)
cycleFanState()	213			HVAC_FAN	Momentary Function Channel: Cycle the fan state when channel is activated
cycleHumidifyState()	217			HVAC_HUMIDIFY_STATE	Momentary Function Channel: Cycle the humidify state when channel is activated
cycleHVACState()	218			HVAC_STATE	Momentary Function Channel: Cycle the HVAC state when channel is activated
decrementCoolSetpoint()	141			HVAC_COOL_DN	Momentary Function Channel: Decrement the cool setpoint when channel is activated
decrementDehumidifySetpoint()	151			HVAC_DEHUMIDIFY_DN	Momentary Function Channel: Decrement the dehumidify setpoint when channel is activated
decrementHeatSetpoint()	144			HVAC_HEAT_DN	Momentary Function Channel: Decrement the heat setpoint when channel is activated
decrementHumidifySetpoint()	149			HVAC_HUMIDIFY_DN	Momentary Function Channel: Decrement the humidify setpoint when channel is activated
getHumidifyState()			?HVACHUMID		Query for the humidify state, responds with HVACHUMID-<state> where <state> is OFF, HUMIDIFY, DEHUMIDIFY, AUTO
getHumidifyStatus()			?HVACHUMIDSTATUS		Query for the humidify status, responds with HVACHUMIDSTATUS-<status> where <status> is OFF, HUMIDIFY, DEHUMIDIFY
getHVACComponentAddress (index)			?HVACADDR-<index>		Query for the address of the thermostat at index <index>, responds with HVACADDR-<index>,<hvacAddress>
getHVACComponentIndex (hvacAddress)			?HVACIDX-<hvacAddress>		Query for the index of the thermostat with address <hvacAddress>, responds with HVACADDR-<index>,<hvacAddress>
getTemperatureScale()			?HVACSCALE		Query for the temperature scale, responds with HVACSCALE-<scale> where <scale> is FAHRENHEIT, CELSIUS
incrementCoolSetpoint()	140			HVAC_COOL_UP	Momentary Function Channel: Increment the cool setpoint when channel is activated
incrementDehumidifySetpoint()	150			HVAC_DEHUMIDIFY_UP	Momentary Function Channel: Increment the dehumidify setpoint when channel is activated
incrementHeatSetpoint()	143			HVAC_HEAT_UP	Momentary Function Channel: Increment the heat setpoint when channel is activated
incrementHumidifySetpoint()	148			HVAC_HUMIDIFY_UP	Momentary Function Channel: Increment the humidify setpoint when channel is activated

Component Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
`removeHVACComponent (hvacAddress)`			`HVACREMOVEADDR-<hvacAddress>`		Remove the thermostat with address <hvacAddress>, where <hvacAddress> is a thermostat address
`removeHVACComponent(index)`			`HVACREMOVEIDX-<index>`		Remove the thermostat at index <index>, where <index> is 1-x and x is the maximum supported thermostat index (see module documentation)
`setCoolSetpoint(setpoint)`		31		HVAC_COOL_LVL	Set the cool setpoint, value is in degrees C or F depending on temperature scale
`setDehumidifySetpoint (setpoint)`		38		HVAC_DEHUMIDIFY_LVL	Set the dehumidify setpoint, value is in percent
`setFanState(AUTO)`	215			HVAC_FAN_AUTO	Momentary Function Channel: Fan state is auto while channel is active
`setFanState(ON)`	214			HVAC_FAN_ON	Momentary Function Channel: Fan state is on while channel is active
`setHeatSetpoint(setpoint)`		32		HVAC_HEAT_LVL	Set the heat setpoint, value is in degrees C or F depending on temperature scale
`setHoldOn(state)`	211			HVAC_HOLD_ON	Discrete Function Channel: Thermostat hold mode is on while channel is active
`setHumidifySetpoint(setpoint)`		37		HVAC_HUMIDIFY_LVL	Set the humidify setpoint, value is in percent
`setHumidifyState(AUTO)`	228			HVAC_HUMIDIFY_AUTO	Momentary Function Channel: Humidity state is auto while channel is active
`setHumidifyState(DEHUMIDIFY)`	229			HVAC_DEHUMIDIFY	Momentary Function Channel: Humidity state is dehumidify while channel is active
`setHumidifyState(hs)`			`HVACHUMID-<hs>`		Set the humidify state, where <hs> is OFF,HUMIDIFY,DEHUMIDIFY,AUTO
`setHumidifyState(HUMIDIFY)`	230			HVAC_HUMIDIFY	Momentary Function Channel: Humidity state is humidify while channel is active
`setHumidifyState(OFF)`	231			HVAC_HUMIDIFY_OFF	Momentary Function Channel: Humidity state is off while channel is active
`setHVACState(AUTO)`	219			HVAC_AUTO	Momentary Function Channel: HVAC state is auto while channel is active
`setHVACState(COOL)`	220			HVAC_COOL	Momentary Function Channel: HVAC state is cool while channel is active
`setHVACState(EMERGENCY_HEAT)`	223			HVAC_EHEAT	Momentary Function Channel: HVAC state is emergency heat while channel is active
`setHVACState(HEAT)`	221			HVAC_HEAT	Momentary Function Channel: HVAC state is heat while channel is active
`setHVACState(OFF)`	222			HVAC_OFF	Momentary Function Channel: HVAC state is off while channel is active
`setLockOn(state)`	212			HVAC_LOCK_ON	Discrete Function Channel: Thermostat is locked while channel is active
`setTemperatureScale(ts)`			`HVACSCALE-<ts>`		Set the temperature scale, where <ts> is FAHRENHEIT,CELSIUS

Listener					
Name: HVAC Listener					
Interface: IHVACComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processCoolSetpointEvent		31		HVAC_COOL_LVL	Cool setpoint changed, value is in degrees C or F depending on temperature scale
processDehumidifySetpointEvent		38		HVAC_DEHUMIDIFY_LVL	Dehumidify setpoint changed, value is in percent
processFanStatusEvent	216			HVAC_FAN_STATUS_FB	Feedback Channel: Fan status is on when channel is on
processFanStateEvent	215			HVAC_FAN_AUTO_FB	Feedback Channel: Fan state is Auto while channel is on
processFanStateEvent	214			HVAC_FAN_ON_FB	Feedback Channel: Fan state is on while channel is on
processHeatSetpointEvent		32		HVAC_HEAT_LVL	Heat setpoint changed, value is in degrees C or F depending on temperature scale
processHoldEvent	211			HVAC_HOLD_FB	Feedback Channel: Thermostat hold mode is on while channel is on
processHumidifySetpointEvent		37		HVAC_HUMIDIFY_LVL	Humidify setpoint changed, value is in percent
processHumidifyStateEvent	228			HVAC_HUMIDIFY_AUTO_FB	Feedback Channel: Humidity state change (see chart below)
processHumidifyStateEvent	229			HVAC_DEHUMIDIFY_FB	Feedback Channel: Humidity state change (see chart below)
processHumidifyStateEvent	230			HVAC_HUMIDIFY_FB	Feedback Channel: Humidity state change (see chart below)
processHumidifyStateEvent			HVACHUMID-<state>		Humidify state changed, <state> is OFF,HUMIDIFY,DEHUMIDIFY,AUTO
processHumidifyStateEvent	231			HVAC_HUMIDIFY_OFF_FB	Feedback Channel: Humidity state change (see chart below)
processHumidifyStatusEvent	232			HVAC_DEHUMIDIFING_FB	Feedback Channel: Humidity status change (see chart below)
processHumidifyStatusEvent	233			HVAC_HUMIDIFING_FB	Feedback Channel: Humidity status change (see chart below)
processHumidifyStatusEvent			HVACHUMIDSTATUS-<status>		Humidify status changed, <status> is OFF,HUMIDIFY,DEHUMIDIFY
processHVACStateEvent	219			HVAC_AUTO_FB	Feedback Channel: HVAC state change (see chart below)
processHVACStateEvent	220			HVAC_COOL_FB	Feedback Channel: HVAC state change (see chart below)
processHVACStateEvent	223			HVAC_EHEAT_FB	Feedback Channel: HVAC state change (see chart below)
processHVACStateEvent	221			HVAC_HEAT_FB	Feedback Channel: HVAC state change (see chart below)
processHVACStateEvent	222			HVAC_OFF_FB	Feedback Channel: HVAC state change (see chart below)
processHVACStatusEvent	224			HVAC_COOLING_FB	Feedback Channel: HVAC status change (see chart below)
processHVACStatusEvent	226			HVAC_COOLING2_FB	Feedback Channel: HVAC status change (see chart below)
processHVACStatusEvent	227			HVAC_EHEATING_FB	Feedback Channel: HVAC status change (see chart below)
processHVACStatusEvent	225			HVAC_HEATING_FB	Feedback Channel: HVAC status change (see chart below)
processIndoorHumidityEvent		35		INDOOR_HUMID_LVL	Indoor humidity changed, value is in percent
processIndoorTemperatureEvent		33		INDOOR_TEMP_LVL	Indoor temperature changed, value is in degrees C or F depending on temperature scale
processLockEvent	212			HVAC_LOCK_FB	Feedback Channel: Thermostat is locked while channel is on
processOutdoorHumidityEvent		36		OUTDOOR_HUMID_LVL	Outdoor humidity changed, value is in percent

Listener Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processOutdoorTemperatureEvent		34		OUTDOOR_TEMP_LVL	Outdoor temperature changed, value is in degrees C or F depending on temperature scale
processTemperatureScaleEvent			HVACSCALE-<scale>		HVAC scale changed, <scale> is FAHRENHEIT,CELSIUS

HVAC Listener State Charts

processFanStateEvent		
State	**Channel 214**	**Channel 215**
ON	ON	OFF
AUTO	OFF	ON

processFanStatusEvent	
State	**Channel 216**
OFF	OFF
ON	ON

processHumidifyStateEvent				
State	**Channel 228**	**Channel 229**	**Channel 230**	**Channel 231**
AUTO	ON	OFF	OFF	OFF
DEHUMIDIFY	OFF	ON	OFF	OFF
HUMIDIFY	OFF	OFF	ON	OFF
OFF	OFF	OFF	OFF	ON

processHumidifyStatusEvent		
State	**Channel 232**	**Channel 233**
OFF	OFF	OFF
DEHUMIDIFY	ON	OFF
HUMIDIFY	OFF	ON

processHVACStateEvent					
State	**Channel 219**	**Channel 220**	**Channel 221**	**Channel 222**	**Channel 223**
AUTO	ON	OFF	OFF	OFF	OFF
COOL	OFF	ON	OFF	OFF	OFF
HEAT	OFF	OFF	ON	OFF	OFF
OFF	OFF	OFF	OFF	ON	OFF
EMERGENCY_HEAT	OFF	OFF	OFF	OFF	ON

processHVACStatusEvent				
State	**Channel 224**	**Channel 225**	**Channel 226**	**Channel 227**
OFF	OFF	OFF	OFF	OFF
COOL	ON	OFF	OFF	OFF
HEAT	OFF	ON	OFF	OFF
COOL_2	OFF	OFF	ON	OFF
EMERGENCY_HEAT	OFF	OFF	OFF	ON

IO Device

Component					
Name: IO Device					
Interface: IIODeviceComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getIOChannelCount()`			`?IOCHANNELCOUNT`		Query for the number of channels on an IO device, responds with IOCHANNELCOUNT
`getIOChannelDirection(io-chan)`			`?IOCHANNELDIRECTION-<channel>`		Query for direction of the I/O channel, where <io-chan> is the integer channel number, responds with IOCHANNELDIRECTION
`getIOChannelInputSense(io-chan)`			`?IOCHANNELINPUTSENSE-<channel>`		Query for input sense of the I/O channel, where <io-chan> is the integer channel number, responds with IOCHANNELINPUTSENSE
`getIOChannelState(io-chan)`			`?IOCHANNELSTATE-<channel>`		Query for state of the I/O channel, where <io-chan> is the integer channel number, responds with IOCHANNELSTATE
`setIOChannelDirection(io-chan, io-dir)`			`IOCHANNELDIRECTION-<channel>,<direction>`		Sets the I/O channel direction, where <io-chan> is the integer channel number and <io-dir> is INPUT or OUTPUT
`setIOChannelInputSense(io-chan, io-sense)`			`IOCHANNELINPUTSENSE-<channel>,<sense>`		Sets the I/O channel input sense, where <io-chan> is the integer channel number and <io-sense> is HIGH or LOW
`setIOChannelState(io-chan, io-state)`			`IOCHANNELSTATE-<channel>, <state>`		Sets the I/O channel state, where <io-chan> is the integer channel number and <io-state> is ON or OFF

Listener					
Name: IO Device Listener					
Interface: IIODeviceComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processIOChannelCountEvent`			`IOCHANNELCOUNT-<count>`		Response to ?IOCHANNELCOUNT, where <count> is an integer value
`processIOChannelDirectionEvent`			`IOCHANNELDIRECTION-<channel>,<direction>`		Response to ?IOCHANNELDIRECTION, where <io-chan> is the integer channel number and <io-dir> is INPUT or OUTPUT
`processIOChannelInputSenseEvent`			`IOCHANNELINPUTSENSE-<channel>,<sense>`		Response to ?IOCHANNELINPUTSENSE, where <io-chan> is the integer channel number and <io-sense> is HIGH or LOW
`processIOChannelStateEvent`			`IOCHANNELSTATE-<channel>,<state>`		Response to ?IOCHANNELSTATE, where <io-chan> is the integer channel number and <io-state> is ON or OFF

Keypad

Component					
Name: Keypad					
Interface: IKeypadComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`addKeypadComponent (index,keypadAddress)`			`KEYPADADD-<index>,<keypadAddress>`		Add a keypad address at a given index, where <index> is 1-x and <keypadaddr> is a keypad address and x is the maximum supported keypad index (see module documentation)
`getKeypadComponentAddress (index)`			`?KEYPADADDR-<index>`		Query for the address of the keypad at index <index>, responds with KEYPADADDR-<index>,<keypadaddr>
`getKeypadComponentIndex (keypadAddress)`			`?KEYPADIDX-<keypadaddr>`		Query for the index of the keypad with address <keypadaddr>, responds with KEYPADADDR-<index>,<keypadaddr>
`removeKeypadComponent (index)`			`KEYPADREMOVEIDX-<index>`		Remove the keypad at index <index>, where <index> is 1-x and x is the maximum supported keypad index (see module documentation)
`removeKeypadComponent (keypadAddress)`			`KEYPADREMOVEADDR-<keypadAddress>`		Remove the keypad with address <keypadaddr>, where <keypadaddr> is a keypad address
`setButtonState(btnNum,bs)`			`KEYPADBTN-<btnAddr>,<state>`		Set the state of a keypad button <btn> for the keypad at index/port, where <state> is CLICK or DOUBLE_CLICK
`setButtonStatus(btnNum,bs)`	<btn>				Discrete Function Channel: Set the status of a keypad button <btn> for the keypad at index/port (see state chart)
`setButtonStatus(btnNum,bs)`	<btn>+100				Discrete Function Channel: Set the status of a keypad button <btn> for the keypad at index/port (see state chart)

Listener					
Name: Keypad Listener					
Interface: IKeypadComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processButtonStateEvent`			`KEYPADBTN-<btn>,<state>`		Button State changed. <btn> is button number and <state> is CLICK or DOUBLE_CLICK
`processButtonStateEvent`	<btn>				Input Function Channel: Button state is Released when BUTTON_EVENT-RELEASE is received, where <btn> is the button/channel number
`processButtonStateEvent`	<btn>				Input Function Channel: Button state is Pushed when BUTTON_EVENT-PUSH is received, where <btn> is the button/channel number
`processButtonStatusEvent`	<btn>+100				Feedback Channel: Button status on BLINK while channel <btn> is on and channel <btn>+100 is on, where <btn> is the button/channel number (see chart below)
`processButtonStatusEvent`	<btn>				Feedback Channel: Button status on ON while channel <btn> is on and channel <btn>+100 is off, where <btn> is the button/channel number (see chart below)

Keypad Listener State Charts

processButtonStateEvent	
State	**Channel <btn>**
RELEASE	OFF
PUSH	ON

processButtonStatusEvent		
State	**Channel <btn>**	**Channel <btn>+100**
OFF	OFF	OFF
ON	ON	OFF
BLINK	ON	ON
Note: All Keypad commands include an index. This index is used to obtain the keypad component and the function is called on that component. The range for channel '<btn>' is 1 to 100; the corresponding range for channel '<btn> + 100' is 101 to 200. Channels above 200 are passed to the module for advanced processing.		

KeypadSystem

Component					
Name: KeypadSystem					
Interface: IKeypadSystemComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getKeypadSystemButtonState (keypadAddress)`			`?KEYPADSYSTEMBUTTONSTATE-<keypadAddress>`		Query the state of a button, at the given button <address>. Responds with "KEYPADSYSTEMBUTTONSTATE-<state>", where <state> is CLICK, DOUBLE_CLICK, PUSH, or RELEASE.
`getKeypadSystemButtonStatus (keypadAddress)`			`?KEYPADSYSTEMBUTTONSTATUS-<keypadAddress>`		Query the status of a button, at the given button <address>. Responds with "KEYPADSYSTEMBUTTONSTATUS-<address>,<status>" where <status> is ON, OFF, or BLINK.
`setKeypadSystemButtonState (keypadAddress,buttonState)`			`KEYPADSYSTEMBUTTONSTATE-<keypadAddress>,<buttonState>`		Set the state of a button, at the given button <address>, for the keypad at index/port, to a button state of CLICK, DOUBLE_CLICK, PUSH, or RELEASE.
`setKeypadSystemButtonStatus (keypadAddress,buttonStatus)`			`KEYPADSYSTEMBUTTONSTATUS-<keypadAddress>,<buttonStatus>`		Set the status of a button, at the given button <address>, for the keypad at index/port. to a button status of ON, OFF, or BLINK

Listener					
Name: KeypadSystem Listener					
Interface: IKeypadSystemComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processKeypadSystemButtonStateEvent`			`KEYPADSYSTEMBUTTONSTATE-<keypadAddress>,<buttonState>`		State of a button changed at the given button <address>, where <state> is CLICK, DOUBLE_CLICK, PUSH, or RELEASE.
`processKeypadSystemButtonStatusEvent`			`KEYPADSYSTEMBUTTONSTATUS-<keypadAddress>,<buttonStatus>`		Status of a button changed at the given button <address>, where <status> is ON, OFF, or BLINK.

Lamp

Component					
Name: Lamp					
Interface: ILampComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleLampPower()`	9			`POWER`	Momentary Function Channel: Cycle lamp power when channel is activated
`getCoolDownTime()`			`?COOLDOWN`		Query for cool down time, responds with COOLDOWN-<time> where <time> is in seconds
`getLampTime()`			`?LAMPTIME`		Query for lamp time, responds with LAMPTIME-<time> where <time> is in hours
`getWarmUpTime()`			`?WARMUP`		Query for warm up time, responds with WARMUP-<time> where <time> is in seconds
`setCoolDownTime(secs)`			`COOLDOWN-<seconds>`		Set cool down time where <time> is in seconds
`setCounterNotificationOn(state)`			`COUNTERNOTIFY-<state>`		Turn counter notification on or off, where <state> is 1 or 0
`setLampPower(OFF)`	28			`PWR_OFF`	Momentary Function Channel: Lamp power is turned off when channel is activated
`setLampPower(ON)`	27			`PWR_ON`	Momentary Function Channel: Lamp power is turned on when channel is activated
`setLampPower(ps)`	255			`POWER_ON`	Discrete Function Channel: Lamp power is on while channel is active
`setLampTime(hours)`			`LAMPTIME-<hours>`		Set lamp time where <time> is in hours
`setWarmUpTime(secs)`			`WARMUP-<seconds>`		Set warm up time where <time> is in seconds

Listener					
Name: Lamp Listener					
Interface: ILampComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processCoolDownCounterEvent`			`COOLING-<time>`		Cool down counter time, <time> is seconds remaining
`processLampPowerEvent`	254			`LAMP_COOLING_FB`	Feedback Channel: Indicates Lamp is cooling and cannot accept commands
`processLampPowerEvent`	255			`LAMP_POWER_FB`	Feedback Channel: Indicates lamp is ON
`processLampPowerEvent`	253			`LAMP_WARMING_FB`	Feedback Channel: Indicates Lamp is warming and cannot accept commands
`processLampTimeEvent`			`LAMPTIME-<time>`		Lamp time, <time> is elapsed hours
`processWarmUpCounterEvent`			`WARMING-<time>`		Warm up counter time, <time> is seconds remaining

Lamp Listener State Charts

processLampPowerEvent		
State	**Channel 253**	**Channel 255**
OFF	OFF	OFF
COOL	ON	OFF
ON	OFF	ON
WARM	ON	ON

Light

Component					
Name: Light					
Interface: ILightComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`addLightComponent (index,lightAddress)`			`LIGHTADD-<index>,<lightAddress>`		Add a light at a given index, where <index> is 1 through x and <address> is a light address and x is the maximum supported light index (see module documentation)
`cycleLight(index)`			`LIGHTSTATE-<index>,TOGGLE`		Cycle the state of a light for the light at index <index>. This command is relevant for light loads and scenes.
`getLightComponentAddress (index)`			`?LIGHTADDR-<index>`		Query for the address of the light at index <index>, responds with LIGHTADDR-<index>,<address>
`getLightComponentIndex (lightAddress)`			`?LIGHTIDX-<address>`		Query for the index of the light with address <address>, responds with LIGHTADDR-<index>,<address>
`getLightLevel(index)`			`?LIGHTLEVEL-<index>`		Query for the level of a light for the light at index <index>, responds with LIGHTLEVEL-<index>,<level> where <level> is 0-255. This command is relevant for light loads only.
`isLightOn(index)`			`?LIGHTSTATE-<index>`		Query for the state of a light for the light at index <index>, responds with LIGHTSTATE-<index>,<state> where <state> is ON or OFF. This command is relevant for light loads, presets, and scenes.
`removeLightComponent (index)`			`LIGHTREMOVEIDX-<index>`		Remove the light at index <index>, where <index> is 1 through x and x is the maximum supported light index (see module documentation)
`removeLightComponent (lightAddress)`			`LIGHTREMOVEADDR-<lightAddress>`		Remove the light with address <address>, where <address> is a light address
`setLightLevel(index,level, seconds)`			`LIGHTLEVEL-<index>,<level>,<seconds>`		Set the level of a light for the light at index <index> in <time>, <level> where <level> is 0-255 and <time> is in seconds. This command is relevant for light loads only.
`setLightLevelRamp (index,state)`			`LIGHTRAMP-<index>,<state>`		Set the ramping state for the light at index <index>, where <state> is UP, DOWN, or STOP.
`setLightOn(index,state)`			`LIGHTSTATE-<index>,<state>`		Set the state of a light for the light at index <index>, where <state> is ON or OFF. This command is relevant for light loads, presets, and scenes.

Listener					
Name: Light Listener					
Interface: ILightComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processLightEvent`			`LIGHTSTATE-<index>,<state>`		State of a light changed for the light at index <index>, <state> is ON or OFF
`processLightLevelEvent`			`LIGHTLEVEL-<index>,<level>`		Level of a light changed for the light at index <index>, <level> where <level> is 0-255
`processLightLevelRampEvent`			`LIGHTRAMP-<index>,<state>`		Light is ramping for the light at index <index>, <state> is UP, DOWN or STOP

LightSystem

Component					
Name: LightSystem					
Interface: ILightSystemComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleLight(address)`			`LIGHTSYSTEMSTATE-<address>,TOGGLE`		Cycle the state of a light, at the given light <address>. This command is relevant for light loads and scenes.
`isLightOn(address)`			`?LIGHTSYSTEMSTATE-<address>`		Query the state of a light, at the given light <address>. Responds with "LIGHTSYSTEMSTATE-<address>,<state>" where <state> is true for ON, or false for OFF. This command is relevant for light loads, presets, and scenes.
`setLight(address,state)`			`LIGHTSYSTEMSTATE-<address>,<lightstate>`		Set the state of a light, at the given light <address>, where <lightstate> is ON or OFF. This command is relevant for light loads, presets, and scenes.
`getLightLevel(address)`			`?LIGHTSYSTEMLEVEL-<address>`		Query the level of a light, at the given light <address>. Responds with "LIGHTSYSTEMLEVEL-<address>,<level>", where <level> is 0-255. This command is relevant for light loads only.
`setLightLevel(address,level)`			`LIGHTSYSTEMLEVEL-<address>,<level>`		Set the level of a light, at the given light <address>, to a given <level> where <level> is 0-255. This command is relevant for light loads only.
`setLightLevel(address,level,time)`			`LIGHTSYSTEMLEVEL-<address>,<level>,<time>`		Set the level of a light, at the given light <address>, to a given <level>, ramped over the specified <time>, where <level> is 0-255 and <time> is in seconds. This command is relevant for light loads only.
`setLightLevelRamp(address,DOWN)`			`LIGHTSYSTEMRAMP-<address>,DOWN`		Ramp Down a light level, at the given light <address>, until LIGHTSYSTEMRAMP-<address>,STOP is sent. This command is relevant for light loads and scenes.
`setLightLevelRamp(address,STOP)`			`LIGHTSYSTEMRAMP-<address>,STOP`		Stop ramping the light level, at the given light <address>. This command is relevant for light loads and scenes.
`setLightLevelRamp(address,UP)`			`LIGHTSYSTEMRAMP-<address>,UP`		Ramp Up a light level, at the given light <address>, until LIGHTSYSTEMRAMP-<address>,STOP is sent. This command is relevant for light loads and scenes.
`setLightOff(address)`			`LIGHTSYSTEMSTATE-<address>,OFF`		Set the state of a light OFF, at the given light <address>. This is implemented in base and should not be overridden. This command is relevant for light loads, presets, and scenes.
`setLightOn(address)`			`LIGHTSYSTEMSTATE-<address>,ON`		Set the state of a light ON, at the given light <address>. This is implemented in base and should not be overridden. This command is relevant for light loads, presets, and scenes.

Listener					
Name: LightSystem Listener					
Interface: ILightSystemComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processLightLevelEvent`			`LIGHTSYSTEMLEVEL-<address>,<level>`		Level of a light changed at the given <address>, where <level> is an integer from 0-255.
`processLightLevelRampEvent`			`LIGHTSYSTEMRAMP-<address>,<control>`		Light is ramping at the given <address>, where ramp <control> is UP, DOWN, or STOP.
`processLightStateEvent`			`LIGHTSYSTEMSTATE-<address>,<state>`		State of a light changed at the given <address>, where <state> is ON or OFF.

Media DB

Component					
Name: Media DB					
Interface: IMediaDBComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`closeSearchDB(searchHandle)`			`MEDIADBCLOSESEARCH-<searchHandle>`		Close the search associated with search key = <key>
`deleteRecord(recordID)`			`MEDIADBDELETE-<recordID>`		Delete media record with record id of <id>
`queryDB(mdbss)`			`MEDIADBNEXT-<searchHandle>`		Request next record (count =1) starting with the next record <position> for the media search associated with search key <key>. If <position> is not present, it is assumed to be the next record in the search result set based on the last MEDIADBNEXT or MEDIADBPREV command. Responds with MEDIADBNEXT-<key>,<count>,<position> where <key> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryDB(mdbss)`			`MEDIADBPREV-<searchHandle>`		Request previous record (count = 1) for the media search associated with search key <key>. Responses with MEDIADBPREV-<key>,1,<position> where <key> is the search key and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryDB(mdbss)`			`MEDIADBREFRESH-<searchHandle>`		Refresh record starting with the last starting position used with a MEDIADBNEXT or MEDIADBPREV command for the media search associated with search key <key>. Responses with MEDIADBNEXT-<key>,<count>,<position> where <key> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryDB(mdbss,count)`			`MEDIADBNEXT-<searchHandle>,<count>`		Request next record (count =1) starting with record <position> for the media search associated with search key <key>. If <position> is not present, it is assumed to be the next record in the search result set based on the last MEDIADBNEXT or MEDIADBPREV command. Responses with MEDIADBNEXT-<key>,<count>,<position> where <key> is the search
`queryDB(mdbss,count)`			`MEDIADBPREV-<searchHandle>,<count>`		Request previous <count> records for the media search associated with search key <key>. Responses with MEDIADBPREV-<key>,<count>,<position> where <key> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryDB(mdbss,count)`			`MEDIADBREFRESH-<searchHandle>, <count>`		Refresh record starting with the last starting position used with a MEDIADBNEXT or MEDIADBPREV command for the media search associated with search key <key>. Responses with MEDIADBNEXT-<key>,<count>,<position> where <key> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryMediaDBProperties (recordID)`			`?MEDIADBPROPS-<recordID>`		Query for all Media Database Properties for a given record where <id> is the record ID, responds with multiple MEDIADBPROP-<id>,<key>,<value>, one for each property, where <id> is the record id of the record from which to retrieve the properties, <key> is the property key and <value> is the property value.

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`queryMediaDBProperty(id,key)`			`?MEDIADBPROP-<recordID>,<keyName>`		Query for one Media Database Property where <id> is the record ID and <key> is the property key to query, responds with MEDIADBPROP-<id>,<key>,<value> where <id> is the record id of the record from which to retrieve the properties, <key> is the property key and <value> is the property value.
`searchDB(sr)`			`MEDIADBSEARCH-<searchHandle>, <search type>=<search string>, RETURN=<return type>`		Search the media database for records with <search type> equal to <search string>. If <search string> is "*", all records are returned. <key> is a search key used in other search operations, such as closeSearchDB(). It can be any string you like, such as a panel device number or internal key that makes sense for your program. All future DB operation associated with this search will reference this key value. <search type> can be ALL, ID, ARTIST, GENRE, TITLE, KEYWORDS, PLAYLIST, BOOKMARK. RETURN= is optional and limits the type of items returned in the result set where <result type> can be ALL, PICTURE, APPLICATION, TRACK, CHAPTER, PLAYLIST, BOOKMARK, DISC, AUDIO, VIDEO, GENRE, ARTIST, STATION.
`setMediaDBProperty (sID,sName,sValue)`			`MEDIADBPROP-<recordID>,<keyName>, <value>`		Set a Media Database Property where <id> is the record id of the record for which to set the properties, <key> is the property key and <value> is the property value.
`updateRecord(rec)`			`MEDIADBUPDATE-<recordID>,<name>, <record type>[,<url>]`		Update media record with record id of <id>, where <name> is the new name and <record type> is the new record type, i.e. PICTURE, APPLICATION, TRACK, CHAPTER, PLAYLIST, BOOKMARK, DISC, AUDIO, VIDEO, GENRE, ARTIST, STATION and <url> is the existing media URL to associate with the new/updated record

Listener					
Name: Media DB Listener					
Interface: IMediaDBComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processCloseEvent`			`MEDIADBCLOSESEARCH-<searchHandle>`		The search associated with search key <key> have been closed. No more operations can be performed against this search.
`processDeleteRecordEvent`			`MEDIADBDELETE-<recordID>,<success>`		Media record with record id of <id> was deleted if <success> is 1, otherwise delete failed.
`processEndOfSetEvent`			`MEDIADBEND-<searchHandle>`		End of search set was reached for search with search key <key>
`processMediaDBPropertiesEvent`			`MEDIADBPROP-<recordID>,<keyName>, <value>`		Media Database property value where <id> is the record ID, <key> is the property key and <value> is the property value. One command is returned for each key.
`processQueryDBEvent`			`MEDIADBRECORD-<key>,<recordID>, <resultNumber>,<name>,<record type>, <url>`		Media record for search with search key <key>. <id> is the record ID, <#> is the record position from 1 to the total number of records, <name> is the item name, <record type> is the record type, which could be PICTURE, APPLICATION, TRACK, CHAPTER, PLAYLIST, BOOKMARK, DISC, AUDIO, VIDEO, GENRE, ARTIST, STATION and <url> is the URL of the media.
`processSearchDBEvent`			`MEDIADBSEARCHRESULT-<searchHandle>, <count>`		Media search results are available for search with search key <key>. <count> is the total count of records. Use MEDIADBNEXT and MEDIADBPREV to get more records.
`processStartOfSetEvent`			`MEDIADBSTART-<searchHandle>`		Beginning of search set was reached for search with search key <key>
`processUpdateRecordEvent`			`MEDIADBUPDATE-<recordID>,<success>`		Media record with record id of <id> was updated if <success> is 1, otherwise update failed.

Media Device

Component					
Name: Media Device					
Interface: IMediaDeviceComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleMediaDeviceRandom()`	124			`MEDIA_RANDOM`	Momentary Function Channel: Cycle random when channel is activated
`cycleMediaDeviceRepeat()`	125			`MEDIA_REPEAT`	Momentary Function Channel: Cycle repeat when channel is activated
`getMediaDeviceSource()`			`?DECODESOURCE`		Query for Media Device Source, responds with DECODESOURCE-<url> where <url> is the URL of the source.
`queryMediaDeviceProperties()`			`?DECODEPROPS`		Query for all Media Device Properties, responds with multiple DECODEPROP-<key>,<value>, one for each property, where <key> is the property key and <value> is the property value.
`queryMediaDeviceProperty(sKeyName)`			`?DECODEPROP-<keyName>`		Query for one Media Device Properties, responds with DECODEPROP-<key>,<value> where <key> is the property key and <value> is the property value.
`setMediaDeviceCounterNotificationOn (state)`			`MEDIACOUNTERNOTIFY-<state>`		Turn media counter notification on or off, where <state> is 1 or 0
`setMediaDeviceRandomState (RANDOM_ALL)`	179			`MEDIA_RANDOM_ALL_ON`	Momentary Function Channel: Random-all is on while channel is active
`setMediaDeviceRandomState (RANDOM_DISC)`	178			`MEDIA_RANDOM_DISC_ON`	Momentary Function Channel: Random-disc is on while channel is active
`setMediaDeviceRandomState (RANDOM_OFF)`	180			`MEDIA_RANDOM_OFF_ON`	Momentary Function Channel: Random-off is on while channel is active
`setMediaDeviceRepeatState (REPEAT_ALL)`	183			`MEDIA_REPEAT_ALL_ON`	Momentary Function Channel: Repeat-all is on while channel is active
`setMediaDeviceRepeatState (REPEAT_DISC)`	181			`MEDIA_REPEAT_DISC_ON`	Momentary Function Channel: Repeat-disc is on while channel is active
`setMediaDeviceRepeatState (REPEAT_OFF)`	184			`MEDIA_REPEAT_OFF_ON`	Momentary Function Channel: Repeat-off is on while channel is active
`setMediaDeviceRepeatState (REPEAT_TRACK)`	182			`MEDIA_REPEAT_TRACK_ON`	Momentary Function Channel: Repeat-track is on while channel is active
`setMediaDeviceSource(cURISource)`			`DECODESOURCE-<url>`		Set Media Device Source where <url> is the URL of the source.

Listener					
Name: Media Device Listener					
Interface: IMediaDeviceComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processMediaDeviceCounterEvent`			`MEDIACOUNTER-<counter>`		Media counter changed, where <counter> is a String in the format [-]hh:mm:ss.ff
`processMediaDevicePropertiesEvent`			`DECODEPROP-<key>,<value>`		Media property value where <key> is the property key and <value> is the property value. One command is returned for each key.
`processMediaDeviceRandomStateEvent`	179			`MEDIA_RANDOM_ALL_FB`	Feedback Channel: Random state change (see chart below)
`processMediaDeviceRandomStateEvent`	178			`MEDIA_RANDOM_DISC_FB`	Feedback Channel: Random state change (see chart below)
`processMediaDeviceRandomStateEvent`	180			`MEDIA_RANDOM_OFF_FB`	Feedback Channel: Random state change (see chart below)
`processMediaDeviceRepeatStateEvent`	183			`MEDIA_REPEAT_ALL_FB`	Feedback Channel: Repeat state change (see chart below)
`processMediaDeviceRepeatStateEvent`	181			`MEDIA_REPEAT_DISC_FB`	Feedback Channel: Repeat state change (see chart below)
`processMediaDeviceRepeatStateEvent`	184			`MEDIA_REPEAT_OFF_FB`	Feedback Channel: Repeat state change (see chart below)
`processMediaDeviceRepeatStateEvent`	182			`MEDIA_REPEAT_TRACK_FB`	Feedback Channel: Repeat state change (see chart below)
`processMediaDeviceSourceInfoEvent`			`DECODESOURCE-<recordID>, <name>,<url>`		Media Device Source changed where <recordID> is the record ID of the source (may be blank), <name> is the name of the source and <url> is the URL of the source.

Media Device Listener State Charts

processMediaDeviceRandomStateEvent			
State	**Channel 178**	**Channel 179**	**Channel 180**
RANDOM_DISC	ON	OFF	OFF
RANDOM_ALL	OFF	ON	OFF
RANDOM_OFF	OFF	OFF	ON

processMediaDeviceRepeatStateEvent				
State	**Channel 181**	**Channel 182**	**Channel 183**	**Channel 184**
REPEAT_DISC	ON	OFF	OFF	OFF
REPEAT_TRACK	OFF	ON	OFF	OFF
REPEAT_ALL	OFF	OFF	ON	OFF
REPEAT_OFF	OFF	OFF	OFF	ON

Menu

Component					
Name: Menu					
Interface: IMenuComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
moveMenuCursor(DOWN)	46			MENU_DN	Momentary Function Channel: Move menu cursor DOWN
moveMenuCursor(DOWN_LEFT)	53			MENU_DN_LT	Momentary Function Channel: Move menu cursor DOWN_LEFT
moveMenuCursor(DOWN_RIGHT)	54			MENU_DN_RT	Momentary Function Channel: Move menu cursor DOWN_RIGHT
moveMenuCursor(LEFT)	47			MENU_LT	Momentary Function Channel: Move menu cursor LEFT
moveMenuCursor(RIGHT)	48			MENU_RT	Momentary Function Channel: Move menu cursor RIGHT
moveMenuCursor(UP)	45			MENU_UP	Momentary Function Channel: Move menu cursor UP
moveMenuCursor(UP_LEFT)	51			MENU_UP_LT	Momentary Function Channel: Move menu cursor UP_LEFT
moveMenuCursor(UP_RIGHT)	52			MENU_UP_RT	Momentary Function Channel: Move menu cursor UP_RIGHT
pressMenuButton(A)			ALPHA-A		Press menu button A
pressMenuButton(AB_REPEAT)	112			MENU_AB_REPEAT	Momentary Function Channel: Press menu button AB_REPEAT
pressMenuButton(ACCEPT)	60			MENU_ACCEPT	Momentary Function Channel: Press menu button ACCEPT to answer an incoming call
pressMenuButton(ADVANCE)	83			MENU_ADVANCE	Momentary Function Channel: Press menu button ADVANCE
pressMenuButton(AM)	79			MENU_AM	Momentary Function Channel: Press menu button AM
pressMenuButton(ANGLE)	117			MENU_ANGLE	Momentary Function Channel: Press menu button ANGLE
pressMenuButton(ASTERISK)	91			MENU_ASTERISK	Momentary Function Channel: Press menu button ASTERISK
pressMenuButton(AUDIO)	118			MENU_AUDIO	Momentary Function Channel: Press menu button AUDIO
pressMenuButton(B)			ALPHA-B		Press menu button B
pressMenuButton(BACK)	81			MENU_BACK	Momentary Function Channel: Press menu button BACK
pressMenuButton(C)			ALPHA-C		Press menu button C
pressMenuButton(CANCEL)	43			MENU_CANCEL	Momentary Function Channel: Press menu button CANCEL
pressMenuButton(CLEAR)	80			MENU_CLEAR	Momentary Function Channel: Press menu button CLEAR
pressMenuButton(COMMA)	94			MENU_COMMA	Momentary Function Channel: Press menu button COMMA
pressMenuButton(CONFERENCE)	96			MENU_CONFERENCE	Momentary Function Channel: Press menu button CONFERENCE
pressMenuButton(CONTINUE)	103			MENU_CONTINUE	Momentary Function Channel: Press menu button CONTINUE
pressMenuButton(D)			ALPHA-D		Press menu button D
pressMenuButton(DASH)	90			MENU_DASH	Momentary Function Channel: Press menu button DASH
pressMenuButton(DECK_A_B)	108			MENU_DECK_A_B	Momentary Function Channel: Press menu button DECK_A_B

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
pressMenuButton(DIAL)	95			MENU_DIAL	Momentary Function Channel: Press menu button DIAL
pressMenuButton(DIGIT_0)	10			DIGIT_0	Momentary Function Channel: Press menu button DIGIT_0
pressMenuButton(DIGIT_1)	11			DIGIT_1	Momentary Function Channel: Press menu button DIGIT_1
pressMenuButton(DIGIT_2)	12			DIGIT_2	Momentary Function Channel: Press menu button DIGIT_2
pressMenuButton(DIGIT_3)	13			DIGIT_3	Momentary Function Channel: Press menu button DIGIT_3
pressMenuButton(DIGIT_4)	14			DIGIT_4	Momentary Function Channel: Press menu button DIGIT_4
pressMenuButton(DIGIT_5)	15			DIGIT_5	Momentary Function Channel: Press menu button DIGIT_5
pressMenuButton(DIGIT_6)	16			DIGIT_6	Momentary Function Channel: Press menu button DIGIT_6
pressMenuButton(DIGIT_7)	17			DIGIT_7	Momentary Function Channel: Press menu button DIGIT_7
pressMenuButton(DIGIT_8)	18			DIGIT_8	Momentary Function Channel: Press menu button DIGIT_8
pressMenuButton(DIGIT_9)	19			DIGIT_9	Momentary Function Channel: Press menu button DIGIT_9
pressMenuButton(DIMMER)	84			MENU_DIMMER	Momentary Function Channel: Press menu button DIMMER
pressMenuButton(DISPLAY)	99			MENU_DISPLAY	Momentary Function Channel: Press menu button DISPLAY
pressMenuButton(DOT)	92			MENU_DOT	Momentary Function Channel: Press menu button DOT
pressMenuButton(E)			ALPHA-E		Press menu button E
pressMenuButton(ENTER)	21			MENU_ENTER	Momentary Function Channel: Press menu button ENTER
pressMenuButton(EXIT)	50			MENU_EXIT	Momentary Function Channel: Press menu button EXIT
pressMenuButton(F)			ALPHA-F		Press menu button F
pressMenuButton(FAVORITES)	102			MENU_FAVORITES	Momentary Function Channel: Press menu button FAVORITES
pressMenuButton(FLASH)	203			MENU_FLASH	Momentary Function Channel: Press menu button FLASH
pressMenuButton(FM)	78			MENU_FM	Momentary Function Channel: Press menu button FM
pressMenuButton(FORWARD)	82			MENU_FORWARD	Momentary Function Channel: Press menu button FORWARD
pressMenuButton(FUNCTION)	65			MENU_FUNCTION	Momentary Function Channel: Press menu button FUNCTION
pressMenuButton(G)			ALPHA-G		Press menu button G
pressMenuButton(GUIDE)	105			MENU_GUIDE	Momentary Function Channel: Press menu button GUIDE
pressMenuButton(H)			ALPHA-H		Press menu button H
pressMenuButton(HELP)	113			MENU_HELP	Momentary Function Channel: Press menu button HELP
pressMenuButton(HOLD)	85			MENU_HOLD	Momentary Function Channel: Press menu button HOLD
pressMenuButton(I)			ALPHA-I		Press menu button I
pressMenuButton(INFO)	101			MENU_INFO	Momentary Function Channel: Press menu button INFO
pressMenuButton(INSTANT_REPLAY)	218			MENU_INSTANT_REPLAY	Momentary Function Channel: Press menu button INSTANT_REPLAY
pressMenuButton(J)			ALPHA-J		Press menu button J
pressMenuButton(K)			ALPHA-K		Press menu button K
pressMenuButton(L)			ALPHA-L		Press menu button L

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
pressMenuButton(LEFT_PAREN)	87			MENU_LT_PAREN	Momentary Function Channel: Press menu button LEFT_PAREN
pressMenuButton(LIST)	86			MENU_LIST	Momentary Function Channel: Press menu button LIST
pressMenuButton(LIVE_TV)	62			MENU_LIVE_TV	Momentary Function Channel: Press menu button LIVE_TV
pressMenuButton(M)			ALPHA-M		Press menu button M
pressMenuButton(MENU)	44			MENU_FUNC	Momentary Function Channel: Press menu button MENU
pressMenuButton(N)			ALPHA-N		Press menu button N
pressMenuButton(O)			ALPHA-O		Press menu button O
pressMenuButton(P)			ALPHA-P		Press menu button P
pressMenuButton(PAGE_DOWN)	107			MENU_PAGE_DN	Momentary Function Channel: Press menu button PAGE_DOWN
pressMenuButton(PAGE_UP)	106			MENU_PAGE_UP	Momentary Function Channel: Press menu button PAGE_UP
pressMenuButton(PLUS_10)	20			MENU_PLUS_10	Momentary Function Channel: Press menu button PLUS_10
pressMenuButton(PLUS_100)	97			MENU_PLUS_100	Momentary Function Channel: Press menu button PLUS_100
pressMenuButton(PLUS_1000)	98			MENU_PLUS_1000	Momentary Function Channel: Press menu button PLUS_1000
pressMenuButton(POUND)	93			MENU_POUND	Momentary Function Channel: Press menu button POUND
pressMenuButton(PPV)	64			MENU_PPV	Momentary Function Channel: Press menu button PPV
pressMenuButton(PREVIEW_INPUT)	129			MENU_PREVIEW_INPUT	Momentary Function Channel: Press menu button PREVIEW_INPUT
pressMenuButton(PROGRAM)	111			MENU_PROGRAM	Momentary Function Channel: Press menu button PROGRAM
pressMenuButton(Q)			ALPHA-Q		Press menu button Q
pressMenuButton(R)			ALPHA-R		Press menu button R
pressMenuButton(RECORD_SPEED)	110			MENU_RECORD_SPEED	Momentary Function Channel: Press menu button RECORD_SPEED
pressMenuButton(REJECT)	61			MENU_REJECT	Momentary Function Channel: Press menu button REJECT to reject an incoming call
pressMenuButton(RESET)	215			MENU_RESET	Momentary Function Channel: Press menu button RESET
pressMenuButton(RETURN)	104			MENU_RETURN	Momentary Function Channel: Press menu button RETURN
pressMenuButton(RIGHT_PAREN)	88			MENU_RT_PAREN	Momentary Function Channel: Press menu button RIGHT_PAREN
pressMenuButton(S)			ALPHA-S		Press menu button S
pressMenuButton(SEND_GRAPHICS)	131			MENU_SEND_GRAPHICS	Momentary Function Channel: Press menu button SEND_GRAPHICS
pressMenuButton(SEND_INPUT)	130			MENU_SEND_INPUT	Momentary Function Channel: Press menu button SEND_INPUT
pressMenuButton(SETUP)	66			MENU_SETUP	Momentary Function Channel: Press menu button SETUP
pressMenuButton(SLEEP)	63			MENU_SLEEP	Momentary Function Channel: Press menu button SLEEP
pressMenuButton(SUBTITLE)	100			MENU_SUBTITLE	Momentary Function Channel: Press menu button SUBTITLE
pressMenuButton(T)			ALPHA-T		Press menu button T
pressMenuButton(THUMBS_DOWN)	58			MENU_THUMBS_DN	Momentary Function Channel: Press menu button THUMBS_DOWN
pressMenuButton(THUMBS_UP)	59			MENU_THUMBS_UP	Momentary Function Channel: Press menu button THUMBS_UP
pressMenuButton(TITLE)	114			MENU_TITLE	Momentary Function Channel: Press menu button TITLE

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
pressMenuButton(TOP_MENU)	115			MENU_TOP_MENU	Momentary Function Channel: Press menu button TOP_MENU
pressMenuButton(TV_VCR)	109			MENU_TV_VCR	Momentary Function Channel: Press menu button TV_VCR
pressMenuButton(U)			ALPHA-U		Press menu button U
pressMenuButton(UNDER_SCORE)	89			MENU_UNDERSCORE	Momentary Function Channel: Press menu button UNDER_SCORE
pressMenuButton(V)			ALPHA-V		Press menu button V
pressMenuButton(VIDEO)	57			MENU_VIDEO	Momentary Function Channel: Press menu button VIDEO
pressMenuButton(W)			ALPHA-W		Press menu button W
pressMenuButton(X)			ALPHA-X		Press menu button X
pressMenuButton(XM)	77			MENU_XM	Momentary Function Channel: Press menu button XM
pressMenuButton(Y)			ALPHA-Y		Press menu button Y
pressMenuButton(Z)			ALPHA-Z		Press menu button Z
pressMenuButton(ZOOM)	116			MENU_ZOOM	Momentary Function Channel: Press menu button ZOOM
selectMenuItem()	49			MENU_SELECT	Momentary Function Channel: Select current menu item

Listener					
Name: Menu Listener					
Interface: IMenuComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Module

Component					
Name: Module					
Interface: IModuleComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getDebugState()`			`?DEBUG`		Query the debug level, responds with DEBUG-<state> where <state> is 1-4 for ERROR, WARNING, DEBUG, INFO
`getFWVersion()`			`?FWVERSION`		Query for the device firmware version, responds with FWVERSION-<version>
`getProperty(key)`			`?PROPERTY-<key>`		Query for the value of property <key>, respond with PROPERTY-<key>,<value>
`getVersion()`			`?VERSION`		Query for the module version, responds with VERSION-<version>
`passThru(buffer)`			`PASSTHRU-<buffer>`		Send a message directly to the device
`reinitialize()`			`REINIT`		Reinitialize communication with the device
`setDebugState(state)`			`DEBUG-<state>`		Set the debug state where <state> is 1-4 for ERROR, WARNING, DEBUG, INFO
`setDeviceDateTime(date)`			`CLOCK-<mm/dd/yyyy> <hh:mm:ss>`		Set the device date/time.
`setPassbackOn(boolean)`			`PASSBACK-<state>`		Set the passback state where <state> is 1 or 0. When passback is on, all response from the device will be passed back to the NetLinx program as a string from the virtual device
`setProperty(key,value)`			`PROPERTY-<key>,<value>`		Set the value of property <key> to <value>

Listener					
Name: Module Listener					
Interface: IModuleComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processDataInitializedEvent`	252			`DATA_INITIALIZED`	Feedback Channel: Module data is synchronized with device while channel is on
`processDebugEvent`			`DEBUG-<state>`		Debug state changed where <state> is 1-4 for ERROR, WARNING, DEBUG, INFO
`processDeviceOnLineEvent`	251			`DEVICE_COMMUNICATING`	Feedback Channel: Communication is established with device while channel is on
`processPassbackEvent`					When passback is on, each string received form the device is sent to the NetLinx program as a string. Use a DATA_EVENT event with a STRING handler to capture the data from the device.

Monitor

Component					
Name: Monitor					
Interface: IMonitorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Monitor Listener					
Interface: IMonitorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Multi Window

Component					
Name: Multi Window					
Interface: IMultiWindowComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleMultiWindowPreset()`	136			`MULTIWIN_PRESET`	Momentary Function Channel: Cycle multi-window preset when channel is activated
`getMultiWindowPreset()`			`?MULTIWINPRESET`		Query for multi-window preset, responds with MULTIWINPRESET-<preset>
`saveMultiWindowPreset(nPresetNum)`			`MULTIWINPRESETSAVE-<preset>`		Save multi-window Preset where <preset> is 1 to x and x is the maximum supported preset (see specific module documentation)
`setMultiWindowPreset(nPresetNum)`			`MULTIWINPRESET-<preset>`		Recall multi-window preset where <preset> is 1 to x and x is the maximum supported preset (see specific module documentation)

Listener					
Name: Multi Window Listener					
Interface: IMultiWindowComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processMultiWindowPresetEvent`			`MULTIWINPRESET-<preset>`		Multi-window preset changed, where <preset> is 1-x and x is the maximum supported preset (see specific module documentation)

Motor

Component					
Name: Motor					
Interface: IMotorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleMotorPreset()`	187			MOTOR_PRESET	Momentary Function Channel: Cycle Motor preset when channel is activated
`getMotorPreset()`			?MOTORPRESET		Query for Motor preset, responds with MOTORPRESET-<preset>
`setMotorDirection(CLOSE)`	5			MOTOR_CLOSE	Momentary Function Channel: Set Motor direction to close, causing motor to move in the CLOSE direction
`setMotorDirection(OPEN)`	4			MOTOR_OPEN	Momentary Function Channel: Set Motor direction to open, causing motor to move in the OPEN direction
`setMotorDirection(STOP)`	2			MOTOR_STOP	Momentary Function Channel: Set Motor direction to stop, causing motor to stop between opened and closed
`setMotorPosition(position)`		6		MOTOR_POS_LVL	Recall Motor position, range is 0-255, 0 is close, 255 is open
`setMotorPreset(preset)`			MOTORPRESET-<preset>		Set Motor preset where <preset> is 1 through x where x is the maximum supported preset (see module documentation)

Listener					
Name: Motor Listener					
Interface: IMotorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processMotorDirectionEvent`	5			MOTOR_CLOSE_FB	Feedback Channel: Motor direction is close, motor is moving to the CLOSE position or is closed
`processMotorDirectionEvent`	4			MOTOR_OPEN_FB	Feedback Channel: Motor direction is open, motor is moving to the OPEN position or is open
`processMotorDirectionEvent`	2			MOTOR_STOP_FB	Feedback Channel: Motor is stopped between opened and closed
`processMotorPositionEvent`		6		MOTOR_POS_LVL	Motor position changed, range is 0-255, 0 is close, 255 is open
`processMotorPresetEvent`			MOTORPRESET-<preset>		Motor preset changed where <preset> is 1 through x where x is the maximum supported preset (see module documentation)

Motor Listener State Charts

processMotorDirectionEvent			
State	**Channel 2**	**Channel 4**	**Channel 5**
STOP	ON	OFF	OFF
OPEN	OFF	ON	OFF
CLOSE	OFF	OFF	ON

Output Stream

Component					
Name: Output Stream					
Interface: IOutputStreamComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`addOutputStreamSink(cURISink)`			`ENCODESINKADD-<url>`		Add an Output Stream Sink where <url> is the URL of the sink to be added.
`getOutputStreamSinks()`			`?ENCODESINK`		Query for Output Stream Sinks, responds with multiple ENCODESINK-<url>, one for each output stream sink, where <url> is the URL of the sink/destination.
`getOutputStreamSource()`			`?ENCODESOURCE`		Query for Output Stream Source, responds with ENCODESOURCE-<url> where <url> is the URL for the source.
`queryOutputStreamProperties()`			`?ENCODEPROPS`		Query for all Output Stream Media Properties, responds with multiple ENCODEPROP-<key>,<value>, one for each property, where <key> is the property key and <value> is the property value.
`queryOutputStreamProperty(sName)`			`?ENCODEPROP-<keyName>`		Query for one Output Stream Media Properties, responds with ENCODEPROP-<key>,<value> where <key> is the property key and <value> is the property value.
`removeOutputStreamSink(cURISink)`			`ENCODESINKREMOVE-<url>`		Remove an Output Stream Sink where <url> is the URL of the sink to be removed.
`setOutputStreamProperty(sName,sValue)`			`ENCODEPROP-<keyName>,<value>`		Set an Output Stream Property where <key> is the property key and <value> is the property value.
`setOutputStreamSource(cURISource)`			`ENCODESOURCE-<url>`		Set Output Stream Source where <url> is the URL for the source.

Listener					
Name: Output Stream Listener					
Interface: IOutputStreamComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processOutputStreamPropertiesEvent`			`ENCODEPROP-<key>,<value>`		Output Stream Property value where <key> is the property key and <value> is the property value. One command is returned for each key.
`processOutputStreamSinkAddEvent`			`ENCODESINKADD-<url>`		Output Stream Sink added where <url> is the URL of the added sink.
`processOutputStreamSinkRemoveEvent`			`ENCODESINKREMOVE-<url>`		Output Stream Sink removed where <url> is the URL of the removed sink.
`processOutputStreamSourceEvent`			`ENCODESOURCE-<url>`		Output Stream Source changed where <url> is the URL for the source.

Phonebook

Component					
Name: Phonebook					
Interface: IPhonebookComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`closeSearchDB(searchHandle)`			`PHONEBOOKCLOSESEARCH-<searchHandle>`		Close the search associated with search key = <searchHandle>
`deleteRecord(recordID)`			`PHONEBOOKDELETE-<recordID>`		Delete phonebook record with index/id of <recordID>
`getPhonebookCapacity()`			`?PHONEBOOKCAPACITY`		Query for the phonebook capacity. Responds with PHONEBOOKCAPACITY-<count> where count is 1 to x and x is the maximum supported phonebook index (see module documentation)
`queryDB(si)`			`PHONEBOOKNEXT-<searchHandle>[,<count>,<position>]`		Request next <count> records starting with record <position> for the phonebook search associated with search key <searchHandle>. If <count> is not present, it is assumed to be 1. If <position> is not present, it is assumed to be the next record in the search result set based on the last PHONEBOOKNEXT or PHONEBOOKPREV command. Responses with PHONEBOOKNEXT-<searchHandle>,<count>,<position> where <searchHandle> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryDB(si)`			`PHONEBOOKPREV-<searchHandle>[,<count>]`		Request previous <count> records for the phonebook search associated with search key <key>. If <count> is not present, it is assumed to be 1. Responses with PHONEBOOKPREV-<key>,<count>,<position> where <key> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`queryDB(si)`			`PHONEBOOKREFRESH-<searchHandle>[,<count>]`		Refresh <count> records starting with the last starting position used with a PHONEBOOKNEXT or PHONEBOOKPREV command for the phonebook search associated with search key <key>. If <count> is not present, it is assumed to be 1. Responses with PHONEBOOKNEXT-<key>,<count>,<position> where <key> is the search key, <count> is the number of records to expect and <position> is the position of the first record to be returned between 1 through the total number of records.
`searchDB(sr)`			`PHONEBOOKSEARCH-<searchHandle>, ID=<id>`		Search the phonebook database for records with ID equal to <id>. If <id> is "*", all records are returned. <searchHandle> is a search key used in other search operations, such as closeSearchDB(). It can be any string you like, such as a panel device number or internal key that makes sense for your program. All future DB operation associated with this search will reference this key value.
`updateRecord(sdr)`			`PHONEBOOKUPDATE-<recordID>, <name>,<number>`		Update phonebook record with index/id of <recordID>, where <name> is the new name and <number> is the new number

Listener					
Name: Phonebook Listener					
Interface: IPhonebookComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processCloseEvent`			`PHONEBOOKCLOSESEARCH-<searchHandle>`		The search associated with search key <searchHandle> have been closed. No more operations can be performed against this search.
`processDeleteRecordEvent`			`PHONEBOOKDELETE-<recordID>,<success>`		Phonebook record with index/id of <recordID> was deleted if <success> is 1, otherwise delete failed.
`processEndOfSetEvent`			`PHONEBOOKEND-<searchHandle>`		End of search set was reached for search with search key <searchHandle>
`processQueryDBEvent`			`PHONEBOOKRECORD-<searchHandle>,<recordID>,<resultNumber>,<name>,<number>`		Phonebook record for search with search key <key>. <recordID> is the record ID, <resultNumber> is the record position from 1 to the total number of records, <name> is the name/label and <number> is the phone number.
`processSearchDBEvent`			`PHONEBOOKSEARCHRESULT-<searchHandle>,<count>`		Phonebook search results are available for search with search key <searchHandle>. <count> is the total count of records. Use PHONEBOOKNEXT and PHONEBOOKPREV to get more records.
`processStartOfSetEvent`			`PHONEBOOKSTART-<searchHandle>`		Beginning of search set was reached for search with search key <searchHandle>
`processUpdateRecordEvent`			`PHONEBOOKUPDATE-<recordID>,<success>`		Phonebook record with index/id of <id> was updated if <success> is 1, otherwise update failed.

Pool Spa

Component					
Name: Pool Spa					
Interface: IPoolSpaComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cyclePoolHeatState()`	123			`POOL_HEAT`	Momentary Function Channel: Cycle Pool heat state when channel is activated
`cycleSpaHeatState()`	124			`SPA_HEAT`	Momentary Function Channel: Cycle Spa heat state when channel is activated
`cycleSpaJets()`	125			`SPA_JETS`	Momentary Function Channel: Cycle Spa jets when channel is activated
`decrementPoolSetpoint()`	153			`POOL_HEAT_DN`	Momentary Function Channel: Pool setpoint is decremented when channel is activated
`decrementSpaSetpoint()`	155			`SPA_HEAT_DN`	Momentary Function Channel: Spa setpoint is decremented when channel is activated
`getPoolSpaTemperatureScale()`			`?POOLSCALE`		Query for the Pool/Spa temperature scale, responds with POOLSCALE-<scale> where <scale> is FAHRENHEIT, CELSIUS
`incrementPoolSetpoint()`	152			`POOL_HEAT_UP`	Momentary Function Channel: Pool setpoint is incremented when channel is activated
`incrementSpaSetpoint()`	154			`SPA_HEAT_UP`	Momentary Function Channel: Spa setpoint is incremented when channel is activated
`isPoolSpaAuxOn(aux)`			`?POOLAUX-<auxNumber>`		Query for Pool/Spa Aux state, responds with POOLAUX-<aux number>, <state> where <state> is 0 (false) or 1 (true)
`setPoolHeatState(HEATER)`	175			`POOL_HEATER`	Momentary Function Channel: Set Pool heat state to heater
`setPoolHeatState(OFF)`	174			`POOL_HEAT_OFF`	Momentary Function Channel: Set Pool heat state to off
`setPoolHeatState(SOLAR)`	176			`POOL_SOLAR`	Momentary Function Channel: Set Pool heat state to solar
`setPoolHeatState(SOLAR_PREFERRED)`	177			`POOL_SOLAR_PREF`	Momentary Function Channel: Set Pool heat state to solar preferred
`setPoolLightOn(state)`	172			`POOL_LIGHT_ON`	Discrete Function Channel: Pool light is on while channel is active
`setPoolPumpOn(state)`	170			`POOL_PUMP_ON`	Discrete Function Channel: Pool pump is on while channel is active
`setPoolSetpoint(nTemperature)`		39		`POOL_HEAT_LVL`	Set Pool setpoint, value is in degrees C or F depending on temperature scale
`setPoolSpaAuxOn(aux,state)`			`POOLAUX-<auxNumber>, <state>`		Set Pool/Spa Aux state where <aux number> is the number of the aux relay 1 to x where x is the maximum supported aux relay and <state> is 1 (on) or 0 (off) (see module documentation)
`setPoolSpaTemperatureScale(ts)`			`POOLSCALE-<ts>`		Set the Pool/Spa temperature scale, where <ts> is FAHRENHEIT, CELSIUS
`setSpaBlowerOn(state)`	186			`SPA_BLOWER_ON`	Discrete Function Channel: Spa blower is on while channel is active
`setSpaHeatState(HEATER)`	179			`SPA_HEATER`	Momentary Function Channel: Set Spa heat state to heater
`setSpaHeatState(OFF)`	178			`SPA_HEAT_OFF`	Momentary Function Channel: Set Spa heat state to off

Component Functions (Cont.)

Name:	Channel:	Level:	Command:	Constant:	Notes:
`setSpaHeatState(SOLAR)`	180			SPA_SOLAR	Momentary Function Channel: Set Spa heat state to solar
`setSpaHeatState(SOLAR_PREFERRED)`	181			SPA_SOLAR_PREF	Momentary Function Channel: Set Spa heat state to solar preferred
`setSpaJets(HIGH)`	185			SPA_JETS_HI	Momentary Function Channel: Set Spa jets to high
`setSpaJets(LOW)`	183			SPA_JETS_LO	Momentary Function Channel: Set Spa jets to low
`setSpaJets(MEDIUM)`	184			SPA_JETS_MED	Momentary Function Channel: Set Spa jets to medium
`setSpaJets(OFF)`	182			SPA_JETS_OFF	Momentary Function Channel: Set Spa jets to off
`setSpaLightOn(state)`	173			SPA_LIGHT_ON	Discrete Function Channel: Spa light is on while channel is active
`setSpaPumpOn(state)`	171			SPA_PUMP_ON	Discrete Function Channel: Spa pump is on while channel is active
`setSpaSetpoint(nTemperature)`		40		SPA_HEAT_LVL	Set Spa setpoint, value is in degrees C or F depending on temperature scale

Listener

Name: Pool Spa Listener

Interface: IPoolSpaComponentListener

Listener Functions:

Name:	Channel:	Level:	Command:	Constant:	Notes:
`processPoolHeatStateEvent`	175			POOL_HEATER_FB	Feedback Channel: Pool heater set to heat state, see state chart
`processPoolHeatStateEvent`	174			POOL_HEAT_OFF_FB	Feedback Channel: Pool heater set to off state, see state chart
`processPoolHeatStateEvent`	176			POOL_SOLAR_FB	Feedback Channel: Pool heater set to solar state, see state chart
`processPoolHeatStateEvent`	177			POOL_SOLAR_PREF_FB	Feedback Channel: Pool heater set to solar preferred state, see state chart
`processPoolHeatStatusEvent`	187			POOL_HEATING	Feedback Channel: Pool heater status is heater, see state chart
`processPoolHeatStatusEvent`	188			POOL_HEATING_SOLAR	Feedback Channel: Pool heater status is solar, see state chart
`processPoolLightOnEvent`	172			POOL_LIGHT_FB	Feedback Channel: Pool light is on while channel is active
`processPoolPumpOnEvent`	170			POOL_PUMP_FB	Feedback Channel: Pool pump is on while channel is active
`processPoolSetpointEvent`		39		POOL_HEAT_LVL	Pool setpoint changed, value is in degrees C or F depending on temperature scale
`processPoolSpaAuxOnEvent`			`POOLAUX-<auxNumber>, <state>`		Pool/Spa Aux state changed where <aux number> is the number of the aux relay 1 to x where x is the maximum supported aux relay and <state> is 1 (on) or 0 (off) (see specific module documentation)
`processPoolSpaOutdoorTemperatureEvent`		34		OUTDOOR_TEMP_LVL	Outdoor air temperature changed, value is in degrees C or F depending on temperature scale
`processPoolSpaTemperatureScaleEvent`			`POOLSCALE-<scale>`		Pool temperature scale changed, <scale> is FAHRENHEIT,CELSIUS
`processPoolTemperatureEvent`		41		POOL_TEMP_LVL	Pool temperature changed, value is in degrees C or F depending on temperature scale
`processSpaBlowerOnEvent`	186			SPA_BLOWER_FB	Feedback Channel: Spa blower is on while channel is active
`processSpaHeatStateEvent`	179			SPA_HEATER_FB	Feedback Channel: Spa heater set to heat state, see state chart
`processSpaHeatStateEvent`	178			SPA_HEAT_OFF_FB	Feedback Channel: Spa heater set to off state, see state chart

Listener Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
processSpaHeatStateEvent	180			SPA_SOLAR_FB	Feedback Channel: Spa heater set to solar state, see state chart
processSpaHeatStateEvent	181			SPA_SOLAR_PREF_FB	Feedback Channel: Spa heater set to solar preferred state, see state chart
processSpaHeatStatusEvent	189			SPA_HEATING	Feedback Channel: Spa heater status is heater, see state chart
processSpaHeatStatusEvent	190			SPA_HEATING_SOLAR	Feedback Channel: Spa heater status is solar, see state chart
processSpaJetsEvent	185			SPA_JETS_HI_FB	Feedback Channel: Spa jets state is high, see state chart
processSpaJetsEvent	183			SPA_JETS_LO_FB	Feedback Channel: Spa jets state is low, see state chart
processSpaJetsEvent	184			SPA_JETS_MED_FB	Feedback Channel: Spa jets state is medium, see state chart
processSpaJetsEvent	182			SPA_JETS_OFF_FB	Feedback Channel: Spa jets state is off, see state chart
processSpaLightOnEvent	173			SPA_LIGHT_FB	Feedback Channel: Spa light is on while channel is active
processSpaPumpOnEvent	171			SPA_PUMP_FB	Feedback Channel: Spa pump is on while channel is active
processSpaSetpointEvent		40		SPA_HEAT_LVL	Spa setpoint changed, value is in degrees C or F depending on temperature scale
processSpaTemperatureEvent		42		SPA_TEMP_LVL	Spa temperature changed, value is in degrees C or F depending on temperature scale

Pool Spa Listener State Charts

processPoolHeatStateEvent				
State	**Channel 174**	**Channel 175**	**Channel 176**	**Channel 177**
OFF	ON	OFF	OFF	OFF
HEATER	OFF	ON	OFF	OFF
SOLAR	OFF	OFF	ON	OFF
SOLAR_PREFERRED	OFF	OFF	OFF	ON

processPoolHeatStatusEvent		
State	**Channel 187**	**Channel 188**
OFF	OFF	OFF
HEATER	ON	OFF
SOLAR	OFF	ON

processSpaHeatStateEvent				
State	**Channel 178**	**Channel 179**	**Channel 180**	**Channel 181**
OFF	ON	OFF	OFF	OFF
HEATER	OFF	ON	OFF	OFF
SOLAR	OFF	OFF	ON	OFF
SOLAR_PREFERRED	OFF	OFF	OFF	ON

processSpaHeatStatusEvent		
State	**Channel 189**	**Channel 190**
OFF	OFF	OFF
HEATER	ON	OFF
SOLAR	OFF	ON

processSpaJetsEvent				
State	**Channel 182**	**Channel 183**	**Channel 184**	**Channel 185**
OFF	ON	OFF	OFF	OFF
LOW	OFF	ON	OFF	OFF
MEDIUM	OFF	OFF	ON	OFF
HIGH	OFF	OFF	OFF	ON

Power

Component					
Name: Power					
Interface: IPowerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cyclePower()`	9			POWER	Momentary Function Channel: Cycle power when channel is activated
`setPower(OFF)`	28			PWR_OFF	Momentary Function Channel: Power is turned off when channel is activated
`setPower(ON)`	27			PWR_ON	Momentary Function Channel: Power is turned on when channel is activated
`setPower(ps)`	255			POWER_ON	Discrete Function Channel: Power is on while channel is active
`setPowerSensor(nld,nIOChan)`			`IOLINK-<nld>,<channel>`		Associate a Power Sensor with the device where <dps> is the DPS in string form, i.e. 17:1:0, and <channel> is the channel on the IO device to which the power sensor is connected.

Listener					
Name: Power Listener					
Interface: IPowerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processPowerEvent`	255			POWER_FB	Feedback Channel: Power state changed, power is on while channel is on

Power Listener State Charts

processPowerEvent	
State	**Channel 255**
OFF	OFF
ON	ON

Pre Amp

Component					
Name: Pre Amp					
Interface: IPreAmpComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`adjustBalance(1)`	164			`BALANCE_UP`	Ramping Channel: Balance is incremented when channel is activated
`adjustBalance(-1)`	165			`BALANCE_DN`	Ramping Channel: Balance is decremented when channel is activated
`adjustBass(1)`	166			`BASS_UP`	Ramping Channel: Bass is incremented when channel is activated
`adjustBass(-1)`	167			`BASS_DN`	Ramping Channel: Bass is decremented when channel is activated
`adjustEqualizer(band,offset)`			`EQUALIZER_OFFSET-(band)=(offset)`		Adjust Equalizer Band, given (band) number is a sequential index, and (offset) adjusts its gain, where (offset) ranges from -255 to 255
`adjustEqualizer(band[],offset[])`			`EQUALIZER_OFFSET-(band)=(offset)[,(band)=(offset)]+`		Adjust Equalizer Bands, given each (band) number, with its (offset) to adjust its gain, where (offset)s range from -255 to 255
`adjustTreble(1)`	168			`TREBLE_UP`	Ramping Channel: Treble is incremented when channel is activated
`adjustTreble(-1)`	169			`TREBLE_DN`	Ramping Channel: Treble is decremented when channel is activated
`cycleLoudness()`	206			`LOUDNESS`	Momentary Function Channel: Cycle loudness when channel is activated
`getEqualizer()`			`?EQUALIZER`		Query Equalizer for all supported (bands) and their (gains). Returns EQUALIZER-(band)=(gain)[,(band)=(gain)]+ where (band) ranges from 1 to 255 and its (gain) ranges from 0 to 255
`getEqualizer(band)`			`?EQUALIZER-(band)`		Query Equalizer for the (gain) of a given (band) number, where (band) number ranges from 1 to 255. Returns a format of EQUALIZER-(band)=(gain) where (gain) ranges from 0 to 255
`getEqualizer(band[])`			`?EQUALIZER-(band)[,(band)]+`		Query Equalizer for all given (bands). Returns EQUALIZER-(band)=(gain)[,(band)=(gain)]+ where (band) ranges from 1 to 255 and its corresponding (gain) ranges from 0 to 255
`getEqualizerBands()`			`?EQUALIZER_BANDS`		Query Equalizer for all its supported (bands), where (band) numbers range from 1 to 255. Returns EQUALIZER_BANDS-(band)[,(band)]+
`getSurroundMode()`			`?SURROUND`		Query surround mode, responds with SURROUND-<mode>, where <mode> is MOVIE,MUSIC,OFF
`nextSurroundMode()`	170			`SURROUND_NEXT`	Momentary Function Channel: Next surround mode is selected when channel is activated
`previousSurroundMode()`	171			`SURROUND_PREV`	Momentary Function Channel: Previous surround mode is selected when channel is activated
`setBalance(balance)`		2		`BALANCE_LVL`	Set balance level, range is -128 to 128, -128 is left and 128 is right
`setBass(bass)`		3		`BASS_LVL`	Set bass level, range is 0-255

Component Functions (Cont.)					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`setEqualizer(band,gain)`			`EQUALIZER-(band)=(gain)`		Set Equalizer bands gain, given a (band) number and (gain), where (band) ranges from 1 to 255, and (gain) ranges from 0 to 255'
`setEqualizer(band[],gain[])`			`EQUALIZER-(band)=(gain)[,(band)=(gain)]+`		Set Equalizer, given each (band) number which ranges from 1 to 255, and its corresponding (gain), which ranges from 0 to 255
`setLoudnessOn(state)`	207			`LOUDNESS_ON`	Discrete Function Channel: Loudness is on while channel is active
`setSurroundMode(sm)`			`SURROUND-<mode>`		Set surround mode, where <mode> is MOVIE,MUSIC,OFF
`setTreble(treble)`		4		`TREBLE_LVL`	Set treble level, range is 0-255

Listener					
Name: Pre Amp Listener					
Interface: IPreAmpComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processBalanceEvent`		2		`BALANCE_LVL`	Balance changed, range is -128 to 128, -128 is left and 128 is right
`processBassEvent`		3		`BASS_LVL`	Bass changed, range is 0-255
`processEqualizerEvent`			`EQUALIZER-(band)=(gain)[,(band)=(gain)]+`		The (gain) changed for the Equalizer at a given (band) number, where (band) ranges from 1 to 255 and its (gain) ranges from 0 to 255
`processLoudnessEvent`	207			`LOUDNESS_FB`	Feedback Channel: Loudness is on if channel is on
`processSurroundModeEvent`			`SURROUND-<mode>`		Surround mode changed, where <mode> is MUSIC,MOVIE,OFF
`processTrebleEvent`		4		`TREBLE_LVL`	Treble changed, range is 0-255

Pre Amp Surround Sound Processor

Component					
Name: Pre Amp Surround Sound Processor					
Interface: IPreAmpSurroundSoundProcessorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Pre Amp Surround Sound Processor Listener					
Interface: IPreAmpSurroundSoundProcessorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Receiver

Component					
Name: Receiver					
Interface: IReceiverComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Receiver Listener					
Interface: IReceiverComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Relay Device

Component					
Name: Relay Device					
Interface: IRelayDeviceComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getRelayChannelCount()`			`?RELAYCHANNELCOUNT`		Query for the number of channels on a relay device, responds with RELAYCHANNELCOUNT
`getRelayChannelState(rly-chan)`			`?RELAYCHANNELSTATE-<channel>`		Query for the relay channel state, where <rly-chan> is the integer channel number, responds with RELAYCHANNELCOUNT
`setRelayChannelState (rly-chan,rly-state)`			`RELAYCHANNELSTATE-<channel>,<state>`		Sets the relay channel state, where <rly-chan> is the integer channel number and <rly-state> is ON or OFF

Listener					
Name: Relay Device Listener					
Interface: IRelayDeviceComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processRelayChannelCountEvent`			`RELAYCHANNELCOUNT-<count>`		Response to ?RELAYCHANNELCOUNT, where <count> is an integer value
`processRelayChannelStateEvent`			`RELAYCHANNELSTATE-<channel>,<state>`		Response to ?RELAYCHANNELSTATE, where <rly-chan> is the integer channel number and <rly-state> is ON or OFF

RFIDSystemComponent

Component					
Name: RFIDSystemComponent					
Interface: IRFIDSystemComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getReaders()`			?READERS		Request a list of all the RF Readers in the configured Reader Network controlled by the Duet Module. Responds with READERS-<readerAddress1>[,<readerAddressN>]* where <readerAddress> is a user configured String associated with a specific RF Reader in an RF Reader Network. The user configures the <readerAddress> and uses it in their NetLinx logic. The Duet Module maps the <readerAddress> to the RF Reader's device specific address. <readerAddress> is a String, the default unconfigured value is the RF Reader's device specific address.
`getReadersByTag(tagId)`			?READERSBYTAG-<tagId>		Request the RF Readers that currently have the given <tagId> acquired. Multiple responses may be sent to support a Tag that is acquired by many RF Readers.The <readerCount> always shows the total number of RF Reader sets in the response, where an RF Reader set consists of the <readerAddress> and <tagSignalStrength>. Responds with READERSBYTAG-<tagId>,<tagName>,<tagInfo>,<tagPercentPower>, <readerCount>,<readerIndex> [,<readerAddress>,<tagSignalStrength>]* where <tagId> is a unique identifier assigned to the Tag by the RF Tag manufacturer, <tagId> is a String; <tagName> is a user configured friendly name associated with the <tagId>. The Duet Module maps <tagId> to the <tagName>. <tagName> is a String, truncated to a 20 character limit. The default unconfigured value is <tagId>; <tagInfo> is a user configured String further defining the associated <tagId>. The Duet Module maps the <tagId> to the <tagInfo>. The <tagInfo> can be used to refine type-based processing. For example the user can configure <tagInfo> to describe assets such as PortableAsset, FixedAsset, Employee, Serviceperson, or Guest. <tagInfo> could also describe the type of Tag as Badge, Keychain, or Tag. <tagInfo> is a String, truncated to a 20 character limit. The default unconfigured value is its <tagId>; <tagPercentPower> is the percent remaining battery power for the <tagId> acquired. <tagPercentPower> is an Integer from 1% to 100%; <readerCount> is the total number of RF Reader sets in the READERSBYTAG response. Possible values are: 0 = no RF Readers currently see this Tag; or positive number N, where there will be N number of RF Reader sets in the response. If the value is 0, no RF Reader sets follow. <readerCount> is an Integer from 0 to the maximum number of RF Readers supported by the Duet Module; <readerIndex> is used to support multiple responses to the ?READERSBYTAG command. <readerIndex> gives the index of the first RF Reader set in the current message. For example, if <readerCount> is 100 the message may be broken up into two responses, each with 50 RF Reader sets. The first response will have <readerCount> = 100 and <readerIndex> = 1.

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getReadersByTag(tagId)` **(Cont.)**					The second response will maintain <readerCount> = 100 and set <readerIndex> = 51. RF Reader sets consist of 2 comma-separated-values: <readerAddress>,<tagSignalStrength>. <readerIndex> is an Integer from 1 to <readerCount>; <readerAddress> is a user configured String associated with a specific RF Reader in an RF Reader Network. The user configures the <readerAddress> and uses it in their NetLinx logic. The Duet Module maps the <readerAddress> to the RF Reader's device specific address. <readerAddress> is a String, the default unconfigured value is the RF Reader's device specific address; and <tagSignalStrength> is the Signal Strength of the associated <tagId> recorded by the RF Reader associated with <readerAddress>. <tagSignalStrength> is an Integer from 0 to 255. <tagSignalStrength> is synonymous with the term RSSI or RF Signal Strength Indicator.
`getReaderStatus(readerAddress)`			`?READERSTATUS-<reader Address>`		Get the reported health status of the RF Reader associated with <readerAddress>. Responds with READERSTATUS-<readerAddress>,<readerStatus>,<errorCount> where <readerAddress> is a user configured String associated with a specific RF Reader in an RF Reader Network. The user configures the <readerAddress> and uses it in their NetLinx logic. The Duet Module maps the <readerAddress> to the RF Reader's device specific address. <readerAddress> is a String, the default unconfigured value is the RF Reader's device specific address; <readerStatus> indicates whether the RF Reader is communicating with the Duet module. Valid TSE values are ONLINE or OFFLINE; <errorCount> reports the number of errors since last reported. This is cleared after it is delivered in the READERSTATUS response.
`getTagInfo(tagId)`			`?TAGINFO-<tagId>`		Request the user configured information associated with the given <tagId>. Responds with TAGINFO-<tagId>,<tagName>,<tagInfo> where <tagId> is a unique identifier assigned to the Tag by the RF Tag manufacturer, <tagId> is a String; <tagName> is a user configured friendly name associated with the <tagId>. The Duet Module maps <tagId> to the <tagName>. <tagName> is a String, truncated to a 20 character limit. The default unconfigured value is <tagId>; <tagInfo> is a user configured String further defining the associated <tagId>. The Duet Module maps the <tagId> to the <tagInfo>. The <tagInfo> can be used to refine type-based processing. For example the user can configure <tagInfo> to describe assets such as PortableAsset, FixedAsset, Employee, Serviceperson, or Guest. <tagInfo> could also describe the type of Tag as Badge, Keychain, or Tag. <tagInfo> is a String, truncated to a 20 character limit. The default unconfigured value is its <tagId>.
`getTagsByReader(readerAddress)`			`?TAGSBYREADER-<readerAddress>`		Request all tags currently acquired by the RF Reader associated with the given <readerAddress>. Multiple responses may be sent to support an RF Reader that senses many tags. The <tagCount> always shows the total number of Tag sets in the response, where a Tag set consists of the <tagId>,<tagName>,<tagInfo>,<tagSignalStrength>, and <tagPercentPower>. <tagIndex> is the index of the first tag in this response. Responds with TAGSBYREADER-<readerAddress>,<tagCount>,<tagIndex> [,<tagId>,<tagName>,<tagInfo>,<tagSignalStrength>,<tagPercentPower>]* where <readerAddress> is a user configured String associated with a specific RF Reader in an RF Reader Network. The user configures the <readerAddress> and uses it in their NetLinx logic.

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getTagsByReader(readerAddress)` (Cont.)					The Duet Module maps the <readerAddress> to the RF Reader's device specific address. <readerAddress> is a String, the default unconfigured value is the RF Reader's device specific address; <tagCount> is the total number of Tag sets in the TAGSBYREADER response. Possible values are: -1 = RF Reader is OFFLINE; 0 = no Tags are currently acquired by the RF Reader; or a positive number N, where there will be N number of RF Tag sets in the response. If the value is -1 or 0, no Tag sets follow. <tagCount> is an Integer from -1 to the number of Tags acquired by the Reader; <tagIndex> is used to support multiple responses to the ?TAGSBYREADER command. <tagIndex> gives the index of the first Tag set in the current message. For example, if <tagCount> is 200 the message may be broken up into two responses, each with 100 Tag sets. The first response will have <tagCount> = 200 and <tagIndex> = 1. The second response will maintain <tagCount> = 200 and set <tagIndex> = 101. Tag sets consist of 5 comma-separated-values: <tagId>,<tagName>,<tagInfo>,<tagSignalStrength>,<tagPercentPower>. <tagIndex> is an Integer from 1 to <tagCount>; <tagSignalStrength> is the Signal Strength of the associated <tagId> recorded by the RF Reader associated with <readerAddress>. <tagSignalStrength> is an Integer from 0 to 255. <tagSignalStrength> is synonymous with the term RSSI or RF Signal Strength Indicator; <tagPercentPower> is the percent remaining battery power for the <tagId> acquired. <tagPercentPower> is an Integer from 1% to 100%.
`setReaderAutoPoll (readerAddress,autoPoll, interval)`			`READERAUTOPOLL-<readerAddress>, <autoPoll>[,<interval>]`		<autoPoll> is a feature of an RF Reader at <readerAddress> that can be enabled (ON) or disabled (OFF). When the AutoPoll TSE is ON, then ?TAGSBYREADER is sent, per the optional time <interval> where <readerAddress> is a user configured String associated with a specific RF Reader in an RF Reader Network. The user configures the <readerAddress> and uses it in their NetLinx logic. The Duet Module maps the <readerAddress> to the RF Reader's device specific address. <readerAddress> is a String, the default unconfigured value is the RF Reader's device specific address.

Listener					
Name: RFIDSystemComponent Listener					
Interface: IRFIDSystemComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processReadersByTagEvent`			`READERSBYTAG-<tagId>, <tagName>,<tagInfo>, <tagPercentPower>, <readerCount>, <readerIndex> [,<readerAddress>, <tagSignalStrength>]*`		Sends READERSBYTAG-<tagId>,<tagName>,<tagInfo>,<tagPercentPower>, <readerCount>,<readerIndex> [,<readerAddress>,<tagSignalStrength>]* where each <readerAddress> currently acquiring <tagId> is represented as a 2 item Reader set: <readerAddress>, <tagSignalStrength>. The Duet Module is responsible for maintaining the RF Readers currently acquiring each Tag, and for mapping the RF Reader device address to the user configured <readerAddress>.

Listener Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
processReadersEvent			READERS-<readerAddress1>[,readerAddressN>]*		Sends READERS-<readerAddress1>[,<readerAddressN>]* as the list of N <readerAddress>s in the configured RF Reader Network controlled by the Duet Module. The Duet Module is responsible for knowing the RF Readers participating in the RF Reader Network under its control, and for mapping the RF Reader device address to the user configured <readerAddress>.
processReaderStatusEvent			READERSTATUS-<readerAddress>,<readerStatus>,<errorCount>		Sends READERSTATUS-<readerAddress>,<readerStatus>,<errorCount> for the RF Reader associated with the given <readerAddress>.
processTagAcquiredEvent			TAGACQUIRED-<readerAddress>,<tagId>,<tagName>,<tagInfo>,<timestamp>,<tagSignalStrength>,<tagPercentPower>		Unsolicited notification event that the RF Reader associated with <readerAddress> has just acquired the signal of the RF Tag associated with <tagId>. The Duet Module is responsible for mapping the RF Reader device address to the user configured <readerAddress>. The Duet Module may choose to add configuration to enable the user to filter out this event. <timestamp> is when the RF Tag signal was acquired. <timestamp> is a system time, formatted typically as a numeric String yyyymmddhhmissnnn (nnn is up to msec system granularity).
processTagButtonEvent			TAGBUTTON-<tagId>,<tagName>,<tagInfo>,<state>		Unsolicited notification event that the button on the Tag associated with <tagId> was pushed or released. If a single button push or release event is received by more than one RF Reader, the Duet Module is responsible for consolidating the report from all RF Readers into a single TAGBUTTON event given to the NetLinx program. <state> is a TSE with values of PUSH or RELEASE.
processTagInfoEvent			TAGINFO-<tagId>,<tagName>,<tagInfo>		Sends TAGINFO-<tagId>,<tagName>,<tagInfo> for the RF Tag associated with the given <tagId>.
processTagLostEvent			TAGLOST-<readerAddress>,<tagId>,<tagName>,<tagInfo>,<lastTimestamp>		Unsolicited notification event that the RF Reader associated with <readerAddress> has just lost the signal of the RF Tag associated with <tagId>. The Duet Module is responsible for mapping the RF Reader device address to the user configured <readerAddress>. The Duet Module may choose to add configuration to enable the user to filter out this event. <lastTimestamp> is when the last valid <tagSignalStrength> was acquired for the RF Tag associated with <tagId> by this Reader. <lastTimestamp> is a system timestamp.
processTagsByReaderEvent			TAGSBYREADER-<readerAddress>,<tagCount>,<tagIndex>[,<tagId>,<tagName>,<tagInfo>,<tagSignalStrength>,<tagPercentPower>]*		Sends TAGSBYREADER-<readerAddress>,<tagCount>,<tagIndex> [,<tagId>,<tagName>,<tagInfo>,<tagSignalStrength>, <tagPercentPower>]* where each <tagId>, currently acquired by the RF Reader associated with <readerAddress>, is represented as a 5 item Tag set: <tagId>,<tagName>,<tagInfo>,<tagSignalStrength>,<tagPercentPower>. The Duet Module is responsible for maintaining the Tags currently acquired by each RF Reader, and for mapping the RF device address to the user configured <readerAddress>.
processTagSignalStrengthEvent			TAGSIGNALSTRENGTH-<readerAddress>,<tagId>,<tagName>,<tagInfo>,<timestamp>,<tagSignalStrength>,<tagPercentPower>		Unsolicited notification event that the RF Reader associated with <readerAddress> received a <tagSignalStrength> for the RF Tag associated with <tagId> that is different from its last known <tagSignalStrength>. The Duet Module is responsible for maintaining the last signal strength reported by each RF Reader for each Tag it has acquired. The Duet Module can then check the stored signal strength against the latest reported signal strength to see if it has changed. The Duet Module may choose to add configuration to enable the user to filter out this event.

Security System

Component					
Name: Security System					
Interface: ISecuritySystemComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getPointState(secPoint)`			`?SECPOINTSTATE-<point>`		Query for Point State, responds with SECPOINTSTATE-<point>,<state> where <point> is 1 to the maximum point as returned by getPointCount() (see specific module documentation) and <state> is BYPASS or ACTIVE
`getPointStatus(secPoint)`			`?SECPOINTSTATUS-<point>`		Query for Point Status, responds with SECPOINTSTATUS-<point>,<status> where <point> is 1 to the maximum point as returned by getPointCount() (see specific module documentation) and <status> is ACTIVE, FAULT or BYPASSED
`getSecurityState()`			`?SECSTATE`		Query for Security State, responds with SECSTATE-<state> where <state> is ARM_HOME, ARM, ARM_HOME_NOW, ARM_NOW, DISARM, FIRE, PANIC, POLICE, MEDICAL or NONE
`getSecurityStatus()`			`?SECSTATUS`		Query for Security Status, responds with SECSTATUS-<status> where <status> is DISARMED, ARMED_HOME, ARMED, ALARM
`isOKToArm()`			`?SECARMABLE`		Query for Security arm-able status, responds with SECARMABLE-<status> where <status> is 1 or 0
`setPointState(secPoint,ps,pw)`			`SECPOINTSTATE-<point>,<state>,<password>`		Set Point State where <point> is 1 to the maximum point as returned by getPointCount() (see specific module documentation), <state> is ACTIVE or BYPASS and <password> is the password required to complete the operation (see specific module documentation).
`setSecurityState(ss,password)`			`SECSTATE-<state>,<password>`		Set Security State where <state> is ARM_HOME, ARM, ARM_HOME_NOW, ARM_NOW, DISARM, FIRE, PANIC, POLICE, MEDICAL and <password> is the password required to complete the operation (see module documentation).

Listener					
Name: Security System Listener					
Interface: ISecuritySystemComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processOKToArmEvent`			`SECARMABLE-<status>`		Security arm-able changed, where <status> is 1 or 0, 1 means the security system can be armed.
`processPointStateEvent`			`SECPOINTSTATE-<point>,<state>`		Point State changed, where <point> is 1 to the maximum point as returned by getPointCount() (see specific module documentation) and <state> is BYPASS or ACTIVE
`processPointStatusEvent`			`SECPOINTSTATUS-<point>,<status>`		Point Status changed, where <point> is 1 to the maximum point as returned by getPointCount() (see specific module documentation) and <status> is ACTIVE, FAULT or BYPASSED
`processSecurityStateEvent`			`SECSTATE-<state>`		Security State changed, where <state> is ARM_HOME, ARM, ARM_HOME_NOW, ARM_NOW, DISARM, FIRE, PANIC, POLICE, MEDICAL or NONE
`processSecurityStatusEvent`			`SECSTATUS-<status>`		Security Status changed, where <status> is DISARMED, ARMED_HOME, ARMED, ALARM

Sensor

Component					
Name: Sensor					
Interface: ISensorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Sensor Listener					
Interface: ISensorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processSensorStateEvent`	255			SENSOR_FB	Feedback Channel: Sensor state changed, sensor is on while channel is on
`processSensorValueEvent`		7		SENSOR_VALUE	Value of the sensor changed, range is specific to the sensor type (see specific module documentation)

Sensor Listener State Charts

processSensorStateEvent	
State	**Channel 255**
OFF	OFF
ON	ON

Settop Box

Component					
Name: Settop Box					
Interface: ISettopBoxComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleABSwitch()`	42			CABLE_AB	Momentary Function Channel: Cycle AB switch when channel is activated
`setBSwitchOn(state)`	212			CABLE_B_ON	Discrete Function Channel: AB switch set to B when channel is active

Listener					
Name: Settop Box Listener					
Interface: ISettopBoxComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processBSwitchEvent`	212			CABLE_B_FB	Feedback Channel: AB switch set to B when channel is on

Slide Projector

Component					
Name: Slide Projector					
Interface: ISlideProjectorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getSlide()`			`?SLIDE`		Query for Slide, responds with SLIDE-<slide> where <slide> is 1 to x where x is the maximum support slide (see specific module documentation)
`nextSlide()`	4			`SLIDE_NEXT`	Momentary Function Channel: Slide is advanced to the next slide when channel is activated
`previousSlide()`	5			`SLIDE_PREV`	Momentary Function Channel: Slide is advanced to the previous slide when channel is activated
`setFocusRamp(FAR)`	161			`FOCUS_FAR`	Ramping Channel: Focus is ramped far while channel is active
`setFocusRamp(NEAR)`	160			`FOCUS_NEAR`	Ramping Channel: Focus is ramped near while channel is active
`setSlide(int)`			`SLIDE-<slide>`		Set Slide where <slide> is 1 to x where x is the maximum support slide (see specific module documentation)

Listener					
Name: Slide Projector Listener					
Interface: ISlideProjectorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processFocusRampEvent`	161			`FOCUS_FAR_FB`	Feedback Channel: Focus is ramping far while channel is on
`processFocusRampEvent`	160			`FOCUS_NEAR_FB`	Feedback Channel: Focus is ramping near while channel is on
`processSlideEvent`			`SLIDE-<slide>`		Current Slide changed where <slide> is 1 to x where x is the maximum support slide (see specific module documentation)

Source Select

Component					
Name: Source Select					
Interface: ISourceSelectComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleInputSelect()`			`CYCLESELECT`		Cycles to the next input select which has a different Group number than the current Group number, then responds with INPUTSELECT-<index>
`cycleInputSource()`	196			`SOURCE_CYCLE`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Cycle input source when channel is activated
`getInputCount()`			`?INPUTCOUNT`		Query for the number of inputs
`getInputGroupSelect()`			`?INPUTGROUPSELECT`		*deprecated Input Group Select in favor of Input Select* Query for currently selected input group
`getInputProperties()`			`?INPUTPROPERTIES`		Query input properties for all inputs, responds with INPUTPROPERTIES-<index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName>[;<index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName>]*
`getInputProperty(index)`			`?INPUTPROPERTY-<index>`		Query input properties for single input, responds with INPUTPROPERTY-<index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName>, where <index> is a virtual input number between 1 and <count>, <inputGroup> is the integer value of the virtual input port on the device (mutually exclusive group), <signalType> is a TSE with type values for RGB, SVIDEO, COMPOSITE, COMPONENT, DVI, HDMI, SDI, VGA, AUDIO, LINE, or USB, <deviceLabel> is the label on the device, and <displayName> is the text shown on the device display.
`getInputSelect()`			`?INPUTSELECT`		Gets the current input, where <index> is a virtual input number between 1 and the value returned by ?INPUTCOUNT, responds with INPUTSELECT-<index>
`getInputSource()`			`?INPUT`		*deprecated Input Source in favor of Input Select* Query current input, responds with INPUT-<sourceSelect>,<inputNumber> where <sourceSelect> is RGB, SVIDEO, COMPOSITE, COMPONENT, DVI, HDMI, SDI, VGA, AUDIO, AUXILIARY, CABLE, CAMERA, CD, DVD, FRONT, HDTV, LASERDISC, LINE, MEDIAPLAYER, MINIDISC, PHONO, SATELLITE, TAPE, TUNER, TV, VCR, VIDEO and <inputNumber> is the instance number of the source select
`setInputGroupSelect(group)`			`INPUTGROUPSELECT-<group>`		*deprecated Input Group Select in favor of Input Select* Set input group selection
`setInputSelect(index)`			`INPUTSELECT-<index>`		Sets the current input, where <index> is a virtual input number between 1 and the value returned by ?INPUTCOUNT, responds with INPUTSELECT-<index>

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`setInputSource(AUXILIARY,1)`	39			`SOURCE_AUX1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input AUXILIARY,1 when channel is activated
`setInputSource(CD,1)`	36			`SOURCE_CD1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input CD,1 when channel is activated
`setInputSource(PHONO,1)`	38			`SOURCE_PHONO1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input PHONO,1 when channel is activated
`setInputSource(sourceSelect, inputNumber)`			`INPUT-<sourceSelect>,<inputNumber>`		*deprecated Input Source in favor of Input Select* Set the current input, where <sourceSelect> is RGB, SVIDEO, COMPOSITE, COMPONENT, DVI, HDMI, SDI, VGA, AUDIO, AUXILIARY, CABLE, CAMERA, CD, DVD, FRONT, HDTV, LASERDISC, LINE, MEDIAPLAYER, MINIDISC,PHONO,SATELLITE,TAPE,TUNER,TV,VCR,VIDEO and <inputNumber> is the instance number of the source select
`setInputSource(TAPE,1)`	34			`SOURCE_TAPE1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input TAPE,1 when channel is activated
`setInputSource(TAPE,2)`	35			`SOURCE_TAPE2`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input TAPE,2 when channel is activated
`setInputSource(TUNER,1)`	37			`SOURCE_TUNER1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input TUNER,1 when channel is activated
`setInputSource(TV,1)`	30			`SOURCE_TV1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input TV,1 when channel is activated
`setInputSource(VIDEO,1)`	31			`SOURCE_VIDEO1`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input VIDEO,1 when channel is activated
`setInputSource(VIDEO,2)`	32			`SOURCE_VIDEO2`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input VIDEO,2 when channel is activated
`setInputSource(VIDEO,3)`	33			`SOURCE_VIDEO3`	Momentary Function Channel: *deprecated Input Source in favor of Input Select* Selects input VIDEO,3 when channel is activated

Listener					
Name: Source Select Listener					
Interface: ISourceSelectComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processInputCountEvent`			`INPUTCOUNT-<count>`		Response to ?INPUTCOUNT, where <count> is the integer number of inputs
`processInputGroupSelectEvent`			`INPUTGROUPSELECT-<inputGroup>`		*deprecated Input Group Select in favor of Input Select* Returns integer index of currently selected input group
`processInputPropertiesEvent`			`INPUTPROPERTIES-<index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName>[;<index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName>]*`		Returns comma-separated string containing <index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName> for each input. Where <inputGroup> is the integer value of the virtual input port on the device (mutually exclusive group), <deviceLabel> is the label on the device, <displayName> is the text shown on the device display and <signalType> is RGB, SVIDEO, COMPOSITE, COMPONENT, DVI, HDMI, SDI, VGA, AUDIO, LINE, or USB
`processInputPropertyEvent`			`INPUTPROPERTY-<index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName>`		Returns comma-separated string containing <index>,<inputGroup>,<signalType>,<deviceLabel>,<displayName> for selected input. Where <inputGroup> is the integer value of the virtual input port on the device (mutually exclusive group), <deviceLabel> is the label on the device, <displayName> is the text shown on the device display and <signalType> is RGB, SVIDEO, COMPOSITE, COMPONENT, DVI, HDMI, SDI, VGA, AUDIO, LINE, or USB
`processInputSelectEvent`			`INPUTSELECT-<index>`		Returns integer index of currently selected input
`processInputSourceEvent`			`INPUT-<sourceSelect>,<inputNumber>`		*deprecated Input Source in favor of Input Select* Current input has changed, where <sourceSelect> is RGB, SVIDEO, COMPOSITE, COMPONENT, DVI, HDMI, SDI, VGA, AUDIO, AUXILIARY, CABLE, CAMERA, CD, COMPUTER, DVD, FRONT, HDTV, LASERDISC, LINE, MEDIAPLAYER, MINIDISC, PHONO, SATELLITE, TAPE, TUNER, TV, VCR, VIDEO and <inputNumber> is the instance number of the source select

Switcher

Component					
Name: Switcher					
Interface: ISwitcherComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getInput(sl,output)`			`?INPUT-[<sl>,]<output>`		Query for the input connected to an output, respond with SWITCH-L<sl>I<input>O<output> where <sl> is ALL, VIDEO, or AUDIO and <input> is 0 if there is no connection. If <sl> is not supplied, ALL will be assumed.
`getOutput(sl,input)`			`?OUTPUT-[<sl>,]<input>`		Query for the outputs connected to an input, respond with SWITCH-L<sl>I<input>O<output>, <output>... where <sl> is ALL, VIDEO, or AUDIO and <output> is 0 if there is no connection. If <sl> is not supplied, ALL will be assumed.
`getSwitcherPreset()`			`?SWITCHPRESET`		Query for switcher preset, responds with SWITCHPRESET-<preset>
`saveSwitcherPreset(preset)`			`SWITCHPRESETSAVE-<preset>`		Save switcher preset where <preset> is 1-x and x is the maximum supported preset (see module documentation)
`setSwitcherPreset(preset)`			`SWITCHPRESET-<preset>`		Recall switcher preset where <preset> is 1-x and x is the maximum supported preset (see module documentation)
`switchInputToOutput (ALL,input,output)`			`CI<input>O<output>`		Switch <input> to one or more <output>s for switcher level All. Use <input> 0 for disconnect.
`switchInputToOutput (ALL,input,output[])`			`CI<input>O<output,...>`		Switch <input> to one or more <output>s for switcher level All. Use <input> 0 for disconnect.
`switchInputToOutput (ALL,input[],output[])`			`CI<input,...>O<output,...>`		Switch <input> to one or more <output>s for switcher level All. Use <input> 0 for disconnect.
`switchInputToOutput (AUDIO,input,output)`			`AI<input>O<output>`		Switch <input> to <output> for switcher level Audio. Use <input> 0 for disconnect.
`switchInputToOutput (AUDIO,input,output[])`			`AI<input>O<output,...>`		Switch <input> to one or more <output>s for switcher level Audio. Use <input> 0 for disconnect.
`switchInputToOutput (AUDIO,input[],output[])`			`AI<input,...>O<output,...>`		Switch <input> to one or more <output>s for switcher level Audio. Use <input> 0 for disconnect.
`switchInputToOutput (sl,input,output)`			`CL<sl>I<input>O<output>`		Switch <input> to one or more <output>s where <sl> is ALL, VIDEO, or AUDIO. Use <input> 0 for disconnect.
`switchInputToOutput (sl,input,output[])`			`CL<sl>I<input>O<output,...>`		Switch <input> to one or more <output>s where <sl> is ALL, VIDEO, or AUDIO. Use <input> 0 for disconnect.
`switchInputToOutput (VIDEO,input,output)`			`VI<input>O<output>`		Switch <input> to <output> where <sl> is ALL, VIDEO, or AUDIO. Use <input> 0 for disconnect.
`switchInputToOutput (VIDEO,input,output[])`			`VI<input>O<output,...>`		Switch <input> to one or more <output>s for switcher level Video. Use <input> 0 for disconnect.
`switchInputToOutput (VIDEO,input[],output[])`			`VI<input,...>O<output,...>`		Switch <input> to one or more <output>s for switcher level Video. Use <input> 0 for disconnect.

Listener					
Name: Switcher Listener					
Interface: ISwitcherComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processSwitcherPresetEvent`			`SWITCHPRESET-<preset>`		Switcher preset changed, where <preset> is 1-x and x is the maximum supported preset (see module documentation)
`processSwitchEvent`			`SWITCH-L<sl>I<input> O<output>`		Switch connections changed, where <sl> is ALL, VIDEO, or AUDIO and <input> is 0 if there is no connection.

Tape Transport

Component					
Name: Tape Transport					
Interface: ITapeTransportComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleSearchSpeed()`	119			SEARCH_SPEED	Momentary Function Channel: Cycle search speed when channel is activated
`ejectTape()`	120			EJECT	Momentary Function Channel: Tape is ejected when channel is activated
`resetTapeCounter()`	121			RESET_COUNTER	Momentary Function Channel: Counter is reset when channel is activated
`setTapeCounterNotificationOn state)`			TAPECOUNTERNOTIFY-<state>		Turn counter notification on or off, where <state> is 1 or 0
`setTapeTransport(FF)`	4			FFWD	Momentary Function Channel: Deck is set to fast-forward when the channel is activated
`setTapeTransport(PAUSE)`	3			PAUSE	Momentary Function Channel: Deck is set to pause when the channel is activated
`setTapeTransport(PLAY)`	1			PLAY	Momentary Function Channel: Deck is set to play when the channel is activated
`setTapeTransport(RECORD)`	8			RECORD	Momentary Function Channel: Deck is set to record when the channel is activated
`setTapeTransport(REW)`	5			REW	Momentary Function Channel: Deck is set to rewind when the channel is activated
`setTapeTransport(SEARCH_FWD)`	6			SFWD	Momentary Function Channel: Deck is set to search forward when the channel is activated
`setTapeTransport(SEARCH_REV)`	7			SREV	Momentary Function Channel: Deck is set to search reverse when the channel is activated
`setTapeTransport(SLOW_FWD)`	188			SLOW_FWD	Momentary Function Channel: Deck is set to slow forward when the channel is activated
`setTapeTransport(SLOW_REV)`	189			SLOW_REV	Momentary Function Channel: Deck is set to slow reverse when the channel is activated
`setTapeTransport(STOP)`	2			STOP	Momentary Function Channel: Deck is set to stop when the channel is activated

Listener					
Name: Tape Transport Listener					
Interface: ITapeTransportComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processTapeCounterEvent`			`TAPECOUNTER-<counter>`		Tape counter changed, <counter> is hh:mm:ss.ff
`processTapeLoadedEvent`	122			`TAPE_LOADED_FB`	Feedback Channel: Tape is loaded while channel is on
`processTapeRecordLockedEvent`	123			`RECORD_LOCK_FB`	Feedback Channel: Tape is record locked while channel is on
`processTapeTransportEvent`	244			`FFWD_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	243			`PAUSE_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	241			`PLAY_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	248			`RECORD_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	245			`REW_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	246			`SFWD_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	247			`SREV_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	249			`SLOW_FWD_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	250			`SLOW_REV_FB`	Feedback Channel: Transport state change (see chart below)
`processTapeTransportEvent`	242			`STOP_FB`	Feedback Channel: Transport state change (see chart below)

Tape Transport Listener State Charts

processTapeTransportEvent										
State	**Channel 241**	**Channel 242**	**Channel 243**	**Channel 244**	**Channel 245**	**Channel 246**	**Channel 247**	**Channel 248**	**Channel 249**	**Channel 250**
PLAY	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
STOP	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
PAUSE	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
FF	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
REW	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
SEARCH_FWD	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
SEARCH_REV	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
RECORD	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF
RECORD_PAUSE	OFF	OFF	ON	OFF	OFF	OFF	OFF	ON	OFF	OFF
SLOW_FWD	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
SLOW_REV	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON

Text Keypad

Component					
Name: Text Keypad					
Interface: ITextKeypadComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`clearDisplay()`			`TEXT-`		Clear display. "-" is optional
`setBacklightOn(state)`			`TEXTBACKLIGHT-<state>`		Set BacklightOn where (see specific module documentation)
`setText(line,column,text)`			`TEXT-<line>,<column>,<text>`		Set text starting at a given line and column. Characters will be overwritten as needed based on text. Text will not wrap around to the next line.
`setTextDisplay(text)`			`TEXT-<text>`		Set text starting at line 1 and column 1. Text will wrap around to the next line to fill the display.
`setTextLine(line,text)`			`TEXT-<line>,<text>`		Set text starting at column 1 of the line specified. Text will not wrap around to the next line.

Listener					
Name: Text Keypad Listener					
Interface: ITextKeypadComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Tuner Station

Component					
Name: Tuner Station					
Interface: ITunerStationComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleBand()`	40			`TUNER_BAND`	Momentary Function Channel: Cycle tuner band when channel is activated
`cycleDisplayInfo()`	234			`TUNER_OSD`	Momentary Function Channel: Cycle on-screen or front-panel display info when channel is activated
`cycleStationPresetGroup()`	224			`TUNER_PRESET_GROUP`	Momentary Function Channel: Cycle station preset group/bank when channel is activated
`decrementStation()`	226			`TUNER_STATION_DN`	Momentary Function Channel: Station is decremented when channel is activated
`getBand()`			`?BAND`		Query for band, responds with BAND-<tb> where <tb> is AM, FM, FM_MONO, SATELLITE_RADIO, LONG_WAVE, MEDIUM_WAVE, SHORT_WAVE, TV
`getStation()`			`?XCH`		Query for station, responds with XCH-<station> where <station> is a station string such as "501", "103.7" or "5.1"
`getStationPreset()`			`?TUNERPRESET`		Query for tuner preset, responds with TUNERPRESET-<preset>
`getStationPresetCount()`			`?STATIONPRESETCOUNT`		Query for number of valid station presets, responds with STATIONPRESETCOUNT-<count>
`getStationPresetProperties()`			`?STATIONPRESETPROPERTIES`		Query input properties for all station presets, responds with STATIONPRESETPROPERTIES-<index>,<displayName>,<station> [;<index>,<displayName>,<station>]*
`getStationPresetProperty(index)`			`?STATIONPRESETPROPERTY-<index>`		Query properties for a single station preset, responds with STATIONPRESETPROPERTY-<index>,<displayName>,<station>
`getStationPresetSelect()`			`?STATIONPRESETSELECT`		Query for the currently selected index of valid station presets, responds with STATIONPRESETSELECT-<index>
`getTunerBandProperties()`			`?TUNERBANDPROPERTIES`		Query input properties for all tuner bands, responds with TUNERBANDPROPERTIES-<index>,<displayName>,<tb>[,<index>,<displayName>,<tb>]*
`getTunerBandProperty(index)`			`?TUNERBANDPROPERTY-<index>`		Query properties for a single tuner band, responds with TUNERBANDPROPERTY-<index>,<displayName>,<tb>
`getTunerBandPropertyCount()`			`?TUNERBANDPROPERTYCOUNT`		Query for number of valid tuner bands, responds with TUNERBANDPROPERTYCOUNT-<count>
`getTunerBandSelect()`			`?TUNERBANDSELECT`		Query for the currently selected index of valid tuner bands, responds with TUNERBANDSELECT-<index>

Component Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
getTunerComponentProperties()			?TUNERCOMPONENTPROPERTIES		Query properties for a single tuner component, responds with TUNERCOMPONENTPROPERTIES-<index>,<displayName>,<value>[;<index>,<displayName>,<value>]*
getTunerComponentProperty()			?TUNERCOMPONENTPROPERTY		Query input properties for all inputs, responds with TUNERCOMPONENTPROPERTY-<index>, <displayName>, <value>
getTunerComponentPropertyCount()			?TUNERCOMPONENTPROPERTYCOUNT		Query for number of tuner components, responds with TUNERCOMPONENTPROPERTYCOUNT-<count>
getTunerComponentSelect()			?TUNERCOMPONENTSELECT		Query for the currently selected index of tuner components, responds with TUNERCOMPONENTSELECT-<index>
gotoPreviousStation()	235			TUNER_PREV	Momentary Function Channel: Previous station is selected when channel is activated
incrementStation()	225			TUNER_STATION_UP	Momentary Function Channel: Station is incremented when channel is activated
nextStationPreset()	22			CHAN_UP	Momentary Function Channel: Next station preset is selected when channel is activated
previousStationPreset()	23			CHAN_DN	Momentary Function Channel: Previous station preset is selected when channel is activated
scanStation(BACKWARD)	228			TUNER_SCAN_REV	Momentary Function Channel: Scans for previous station while channel is activate
scanStation(FORWARD)	227			TUNER_SCAN_FWD	Momentary Function Channel: Scans for next station while channel is activate.
seekStation(BACKWARD)	230			TUNER_SEEK_REV	Momentary Function Channel: Seeks for previous station while channel is activate
seekStation(FORWARD)	229			TUNER_SEEK_FWD	Momentary Function Channel: Seeks for next station while channel is activate
setBand(tb)			BAND-<tb>		Set band, where <tb> is AM, FM, FM_MONO, SATELLITE_RADIO, LONG_WAVE, MEDIUM_WAVE, SHORT_WAVE, TV
setStation(station)			XCH-<station>		Set station, where <station> is a station string such as "501", "103.7" or "5.1"
setStationPreset(preset)			TUNERPRESET-<preset>		Recall tuner preset where <preset> is 1-x and x is the maximum supported preset (see specific module documentation)
setStationPresetSelect(index)			STATIONPRESETSELECT-<index>		Set the currently selected index of valid station presets, responds with STATIONPRESETSELECT-<index>
setTunerBandSelect(index)			TUNERBANDSELECT-<index>		Set the currently selected index of valid tuner bands, responds with TUNERBANDSELECT-<index>
setTunerComponentSelect(index)			TUNERCOMPONENTSELECT-<index>		Set the currently selected index of tuner components, responds with TUNERCOMPONENTSELECT-<index>

Listener					
Name: Tuner Station Listener					
Interface: ITunerStationComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processBandEvent`			`BAND-<band>`		Band changed, where <tb> is AM, FM, FM_MONO, SATELLITE_RADIO, LONG_WAVE, MEDIUM_WAVE, SHORT_WAVE, TV
`processStationEvent`			`XCH-<station>`		Station changed, where <station> is a station string such as "501", "103.7" or "5.1"
`processStationPresetCountEvent`			`STATIONPRESETCOUNT-<count>`		Response to ?TUNERCOUNT, where <count> is the integer number of tuner components
`processStationPresetEvent`			`TUNERPRESET-<int>`		Tuner preset changed, where <preset> is 1-x and x is the maximum supported preset (see module documentation)
`processStationPresetPropertiesEvent`			`STATIONPRESETPROPERTIES-<index>, <displayName>,<value> [;<index>,<displayName>, <value>;...]`		Returns the properties for each station preset, where <station> is a string such as "501", "103.7" or "5.1"
`processStationPresetPropertyEvent`			`STATIONPRESETPROPERTY-<index>, <displayName>,<value>`		Returns the properties for a single station preset, where <station> is a string such as "501", "103.7" or "5.1"
`processStationPresetSelectEvent`			`STATIONPRESETSELECT-<index>`		Returns the selected station preset index
`processTunerBandPropertiesEvent`			`TUNERBANDPROPERTIES-<index>, <displayName>,<tunerBand> [;<index>,<displayName>, <tunerBand>;...]`		Returns properties for each tuner band, where <tb> is AM, FM, FM_MONO, SATELLITE_RADIO, LONG_WAVE, MEDIUM_WAVE, SHORT_WAVE, TV
`processTunerBandPropertyCountEvent`			`TUNERBANDPROPERTYCOUNT-<count>`		Returns the number of valid tuner bands
`processTunerBandPropertyEvent`			`TUNERBANDPROPERTY-<index>, <displayName>,<tunerBand>`		Returns properties for a single tuner band, where <tb> is AM, FM, FM_MONO, SATELLITE_RADIO, LONG_WAVE, MEDIUM_WAVE, SHORT_WAVE, TV
`processTunerBandSelectEvent`			`TUNERBANDSELECT-<index>`		Returns the selected tuner band index
`processTunerComponentPropertiesEvent`			`TUNERCOMPONENTPROPERTIES-<index>, <displayName>,<value>[;<index>, <displayName>,<value>;...]`		Returns the properties for each tuner component
`processTunerComponentPropertyCountEvent`			`TUNERCOMPONENTPROPERTYCOUNT`		Returns the integer number of tuner components
`processTunerComponentPropertyEvent`			`TUNERCOMPONENTPROPERTY-<index>, <displayName>,<value>`		Returns the properties for a single tuner component
`processTunerComponentSelectEvent`			`TUNERCOMPONENTSELECT-<index>`		Returns the selected tuner component index

TV

Component					
Name: TV					
Interface: ITVComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: TV Listener					
Interface: ITVComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

UPS

Component					
Name: UPS					
Interface: IUPSComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getAlarms()`			`?UPSALARMS`		Query for alarms, responds with UPSALARMS-<alarms> where <alarms> is a CSV of alarm names
`getBackupProperty backupProperty)`			`?UPSBACKUPPROPERTY-<property>`		Query for the <bu-prop> property, responds with UPSBACKUPPROPERTY-<bu-prop>, <value>
`getBackupStatus()`			`?UPSBACKUPSTATUS`		Query for backup status of UPS, responds with UPSBACKUPSTATUS
`getBackupTimer(backupTimer)`			`?UPSBACKUPTIMER-<timer>`		Query for <bu-timer> value, responds with UPSBACKUPTIMER-<bu-timer>, <value>
`getInputProperty (inputProperty)`			`?UPSINPUTPROPERTY-<property>`		Query for <input-prop>, responds with UPSINPUTPROPERTY-<input-prop>, <value>
`getOutletCount()`			`?UPSOUTLETCOUNT`		Query for number of outlets
`getOutletProperties (outletNumber)`			`?UPSOUTLETPROPERTIES-<outletNumber>`		Query the <outlet#> for all available properties, responds with UPSOUTLETPROPERTIES [<outlet#>,<outlet-prop>,<value>] for each property, CSV format
`getOutletProperty (outletNumber,outletProperty)`			`?UPSOUTLETPROPERTY-<outletNumber>`		Query the <outletNumber> for the <outletProperty>, responds with UPSOUTLETPROPERTY-<outletNumber>,<outletProperty>,<value>
`getOutletState(outlet)`			`?UPSOUTLETSTATE-<outletNumber>`		Query for the state of the specified outlet, responds with UPSOUTLETSTATE
`getState()`			`?UPSSTATE`		Gets the current state of the UPS, responds with UPSSTATE
`getStatus()`			`?UPSSTATUS`		Query for status of UPS, responds with UPSSTATUS
`getTemperatureScale()`			`?UPSTEMPERATURESCALE`		Query for the temperature scale, responds with UPSTEMPERATURESCALE-<tempscale>
`getTestResult()`			`?UPSTESTRESULT`		Query for the results of the last UPSTEST-<test>,<START>, responds with UPSTESTRESULT-<test>,<PASS/FAIL>
`getTestsSupported()`			`?UPSTESTSSUPPORTED`		Query for supported tests, responds with UPSTESTSSUPPORTED-<test_list> where <test_list> is a CSV of supported tests
`setOutletState(outlet,state)`			`UPSOUTLETSTATE-<outletNumber>,<state>`		Sets the outlets state, where <outlet#> is the integer outlet number and <outlet-state> is ON or OFF
`setState(state)`			`UPSSTATE-<state>`		set UPS to the given state
`setState(state,delay)`			`UPSSTATE-<state>,<delay>`		set UPS to the given state in <delay> seconds
`setTemperatureScale (temperatureScale)`			`UPSTEMPERATURESCALE-<temperatureScale>`		Sets the temperature scale
`setTest(test,testAction)`			`UPSTEST-<test>,<testAction>`		Start/Stop/Interrupt the specified <test>, responds with UPSTEST-<test>, <START/STOP/INTERRUPT>

Listener					
Name: UPS Listener					
Interface: IUPSComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processAlarmsEvent`			`UPSALARMS-<alarms>`		<alarms> is a CSV string containing the alarm names
`processBackupPropertyEvent`			`UPSBACKUPPROPERTY-<bu-prop>,<value>`		Data is UPSBackupProperties, <bu-prop> is TSE values as String in upper case (CHARGE, VOLTAGE, CURRENT, REPLACEDATE, or TEMPERATURE)
`processBackupStatusEvent`			`UPSBACKUPSTATUS-<bu-status>`		Data is UPSBackupStatus, <bu-status> is TSE values as String in upper case (OK, LOW, or DEPLETED)
`processBackupTimerEvent`			`UPSBACKUPTIMER-<bu-timer>,<value>`		Data is UPSBackupTimer, <bu-timer> is TSE values as String in upper case (TIMEONBATTERY, TIMETODEPLETED, or TIMETOLOW), <value> is seconds
`processInputPropertyEvent`			`UPSINPUTPROPERTY-<input-prop>,<value>`		Data is UPSInputProperties, <input-prop> is TSE values as String in upper case (DELAYONTIME, DELAYOFFTIME, LINES, FREQUENCY, VOLTAGE, CURRENT, or CAPACITY)
`processIsBackupSupportedEvent`			`UPSBACKUPSUPPORTED-<boolean>`		Response to ?UPSBACKUPSUPPORTED
`processOutletCountEvent`			`UPSOUTLETCOUNT-<count>`		Response to ?UPSOUTLETCOUNTT, where <count> is the integer number of outlets
`processOutletPropertiesEvent`			`UPSOUTLETPROPERTIES-<outletNumber>, <outlet-prop>,<value>[,<outletNumber>, <outlet-prop>,<value>,...]`		Data is UPSOutletProperties, <outlet-prop> is TSE values as String in upper case (OUTPUTLINES, OUTPUTFREQUENCY, OUTPUTVOLTAGE, OUTPUTCURRENT, OUTPUTPOWER, OUTPUTPOWERSOURCE, OUTPUTLOAD, BYPASSLINES, BYPASSFREQUENCY, BYPASSVOLTAGE, BYPASSCURRENT, BYPASSPOWER, BYPASSPOWERSOURCE, or BYPASSLOAD)
`processOutletPropertyEvent`			`UPSOUTLETPROPERTY-<outletNumber>, <outlet-prop>,<value>`		Data is UPSOutletProperties, <outlet-prop> is TSE values as String in upper case (OUTPUTLINES, OUTPUTFREQUENCY, OUTPUTVOLTAGE, OUTPUTCURRENT, OUTPUTPOWER, OUTPUTPOWERSOURCE, OUTPUTLOAD, BYPASSLINES, BYPASSFREQUENCY, BYPASSVOLTAGE, BYPASSCURRENT, BYPASSPOWER, BYPASSPOWERSOURCE, or BYPASSLOAD)
`processOutletStateEvent`			`UPSOUTLETSTATE-<outletNumber>,<state>`		Response to ?UPSOUTLETSTATE-<outlet#>,<outlet-state>, where <outlet-state> is ON or OFF
`processStateEvent`			`UPSSTATE-<state>,<delay>`		Data is UPSState, <ups_state> is TSE values as String in upper case (SHUTDOWN, REBOOT, OUTPUT_ONLY, STARTUP, NORMAL, BATTERY, BYPASS, MANUAL_BYPASS, REDUCING, or BOOSTING)
`processStatusEvent`			`UPSSTATUS-<status>`		Data is UPSStatus, <ups_status> is TSE values as String in upper case (SHUTDOWN, REBOOT, OUTPUT_ONLY, STARTUP, NORMAL, BATTERY, BYPASS, MANUAL_BYPASS, REDUCING, or BOOSTING)
`processTemperatureScaleEvent`			`UPSTEMPERATURESCALE-<temperatureScale>`		Data is TemperatureScale, <tempscale> is TSE values as String in upper case (CELSIUS, OR FAHRENHEIT)
`processTestEvent`			`UPSTEST-<test>,<action>`		Data is TestAction, <action> is TSE values as String in upper case (START, STOP or INTERRUPT)
`processTestResultEvent`			`UPSTESTRESULT-<test>,<result>`		Data is TestResult, <result> is TSE values as String in upper case (PASS or FAIL)
`processTestsSupportedEvent`			`UPSTESTSSUPPORTED-<test_list>`		<test_list> is a CSV string containing each <test> supported

Utility

Component					
Name: Utility					
Interface: IUtilityComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Utility Listener					
Interface: IUtilityComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

VCR

Component					
Name: VCR					
Interface: IVCRComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: VCR Listener					
Interface: IVCRComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Video Conferencer

Component					
Name: Video Conferencer					
Interface: IVideoConferencerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cyclePIP()`	194			`PIP`	Momentary Function Channel: Cycle PIP when channel is activated
`cyclePIPPosition()`	191			`PIP_POS`	Momentary Function Channel: Cycle PIP positions when channel is activated
`cyclePrivacy()`	145			`VCONF_PRIVACY`	Momentary Function Channel: Cycle privacy when channel is activated
`setPIPOn(state)`	195			`PIP_ON`	Discrete Function Channel: PIP is on while channel is active
`setPrivacyOn(state)`	146			`VCONF_PRIVACY_ON`	Discrete Function Channel: Privacy is on while channel is active
`swapPIP()`	193			`PIP_SWAP`	Momentary Function Channel: Swap PIP when channel is activated
`train()`	147			`VCONF_TRAIN`	Momentary Function Channel: Train is executed when the channel is activated

Listener					
Name: Video Conferencer Listener					
Interface: IVideoConferencerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processPIPEvent`	195			`PIP_FB`	Feedback Channel: PIP is on if channel is on
`processPrivacyEvent`	146			`VCONF_PRIVACY_FB`	Feedback Channel: Privacy is muted if channel is on

Video Processor

Component					
Name: Video Processor					
Interface: IVideoProcessorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleVideoProcessorPreset()`	137			VPROC_PRESET	Momentary Function Channel: Cycle video processor preset when channel is activated
`getVideoProcessorPreset()`			?VPROCPRESET		Query for video processor preset, responds with VPROCPRESET-<preset>
`saveVideoProcessorPreset(nPresetNum)`			VPROCPRESETSAVE-<preset>		Save video processor Preset where <preset> is 1 to x and x is the maximum supported preset (see specific module documentation)
`setVideoProcessorPreset(nPresetNum)`			VPROCPRESET-<preset>		Recall video processor preset where <preset> is 1 to x and x is the maximum supported preset (see specific module documentation)

Listener					
Name: Video Processor Listener					
Interface: IVideoProcessorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processVideoProcessorPresetEvent`			VPROCPRESET-<preset>		Video processor preset changed, where <preset> is 1-x and x is the maximum supported preset (see specific module documentation)

Video Projector

Component					
Name: Video Projector					
Interface: IVideoProjectorComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getProjectorPreset()`			`?VPROJPRESET`		Query for projector preset, responds with VPROJPRESET-<preset>
`setProjectorPreset(preset)`			`VPROJPRESET-<preset>`		Recall projector preset where <preset> is 1-x and x is the maximum supported preset (see specific module documentation)

Listener					
Name: Video Projector Listener					
Interface: IVideoProjectorComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processProjectorPresetEvent`			`VPROJPRESET-<int>`		Projector preset changed, where <preset> is 1-x and x is the maximum supported preset (see specific module documentation)

Video Wall

Component					
Name: Video Wall					
Interface: IVideoWallComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getVideoWallConfiguration()`			`?VWALLCONFIG`		Query for video wall configuration, responds with VWALLCONFIG-<config>
`queryVideoWallConfigurationList()`			`?VWALLCONFIGLIST`		Query for the list of available Video Wall Configurations, responds with multiple VWALLCONFIGLIST-<config> commands, one for each valid configuration
`saveVideoWallConfiguration(sConfig)`			`VWALLCONFIGSAVE-<config>`		Save video wall configuration where <config> is the configuration name
`setVideoWallConfiguration(sConfig)`			`VWALLCONFIG-<config>`		Recall video wall configuration where <config> is the configuration name

Listener					
Name: Video Wall Listener					
Interface: IVideoWallComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processVideoWallConfigurationEvent`			`VWALLCONFIG-<config>`		Video wall configuration changed, where <config> is the configuration name
`processVideoWallConfigurationListEvent`			`VWALLCONFIGLIST-<config>`		Response to Video Wall configuration list query, responds with multiple VWALLCONFIGLIST-<config> commands, one for each valid configuration

Volume Controller

Component					
Name: Volume Controller					
Interface: IVolumeControllerComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Listener					
Name: Volume Controller Listener					
Interface: IVolumeControllerComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
Intentionally left blank					

Volume

Component					
Name: Volume					
Interface: IVolumeComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`cycleVolumeMute()`	26			`VOL_MUTE`	Momentary Function Channel: Cycle volume mute when channel is activated
`cycleVolumePreset()`	138			`VOL_PRESET`	Momentary Function Channel: Cycle volume preset when channel is activated
`getVolumePreset()`			`?VOLPRESET`		Query for volume preset, responds with VOLPRESET-<preset>
`saveVolumePreset(preset)`			`VOLPRESETSAVE-<preset>`		Save Volume Preset where <preset> is 1 to x and x is the maximum supported preset (see module documentation)
`setVolume(level)`		1		`VOL_LVL`	Set volume, range is 0-255
`setVolumeMuteOn(state)`	199			`VOL_MUTE_ON`	Discrete Function Channel: Volume mute is on while channel is active
`setVolumePreset(preset)`			`VOLPRESET-<preset>`		Recall volume preset where <preset> is 1 to x and x is the maximum supported preset (see specific module documentation)
`setVolumeRamp(DOWN)`	25			`VOL_DN`	Ramping Channel: Volume is ramped down while channel is active
`setVolumeRamp(UP)`	24			`VOL_UP`	Ramping Channel: Volume is ramped up while channel is active

Listener					
Name: Volume Listener					
Interface: IVolumeComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processVolumeEvent`		1		`VOL_LVL`	Volume changed, range is 0-255
`processVolumeMuteEvent`	199			`VOL_MUTE_FB`	Feedback Channel: Volume is muted if channel is on
`processVolumePresetEvent`			`VOLPRESET-<preset>`		Volume preset changed, where <preset> is 1-x and x is the maximum supported preset (see module documentation)
`processVolumeRampEvent`	25			`VOL_DN_FB`	Feedback Channel: Volume is ramping down while channel is on
`processVolumeRampEvent`	24			`VOL_UP_FB`	Feedback Channel: Volume is ramping up while channel is on

Weather

Component					
Name: Weather					
Interface: IWeatherComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`getChanceOfPrecipitation()`			`?FORECASTCOP`		Query for forecast chance of precipitation changes, responds with FORECASTCOP-<day 1>,<day 2>,etc. Value list will contain one forecast chance of precipitation for each forecast day. The first day is always today. Values are in percent.
`getCurrentCondition()`			`?WEATHERCONDITION`		Query for current conditions, responds with WEATHERCONDITION-<condition>, <condition> will be BLIZZARD, BLOWINGSNOW, CLEAR, CLOUDY, DRIZZLE, DUST, FAIR, FOG, FREEZINGDRIZZLE, FREEZINGRAIN, HAZE, HUMID, ICE, MOSTLYCLOUDY, MOSTLYSUNNY, PARTLYCLOUDY, RAIN, RAINSHOWERS, RAINSNOWMIX, SLEET, SMOKE, SNOW, SNOWFLURRIES, SNOWSHOWERS, SUNNY, THUNDERSTORMS, UNKNOWN, VERYCOLD, WINDY
`getForecastHighTemperature()`			`?FORECASTHIGH`		Query for forecast high temperatures changes, responds with FORECASTHIGH-<day 1>,<day 2>, etc... Value list will contain one forecast high temperature for each forecast day. The first day is always today. Values are in degrees C or F depending on weather scale
`getForecastLowTemperature()`			`?FORECASTLOW`		Query for forecast low temperatures changes, responds with FORECASTLOW-<day 1>,<day 2>, etc... Value list will contain one forecast high temperature for each forecast day. The first day is always today. Values are in degrees C or F depending on weather scale
`getRainfall(dur)`			`?WEATHERRAIN-<duration>`		Query for Rain fall, responds with WEATHERRAIN-<duration>,<value> where <duration> is DAY, WEEK, MONTH, YEAR, YTD and <value> is in inches (Imperial Scale) or cm (Metric Scale).
`getWeatherAlert()`			`?WEATHERALERT`		Query for Weather alert, responds with WEATHERALERT-<alert> where <alert> is a string containing the weather alert
`getWeatherConditions()`			`?FORECASTCONDITION`		Query for forecast conditions, responds with FORECASTCONDITION-<day 1>,<day 2>, etc. Value list will contain one forecast condition for each forecast day. The first day is always today. Values will be BLIZZARD, BLOWINGSNOW, CLEAR, CLOUDY, DRIZZLE, DUST, FAIR, FOG, FREEZINGDRIZZLE, FREEZINGRAIN, HAZE, HUMID, ICE, MOSTLYCLOUDY, MOSTLYSUNNY, PARTLYCLOUDY, RAIN, RAINSHOWERS, RAINSNOWMIX, SLEET, SMOKE, SNOW, SNOWFLURRIES, SNOWSHOWERS, SUNNY, THUNDERSTORMS, UNKNOWN, VERYCOLD, WINDY
`getWeatherScale()`			`?WEATHERSCALE`		Query for the temperature scale, responds with WEATHERSCALE-<scale> where <scale> is IMPERIAL or METRIC
`getWindInfo()`			`?WEATHERWIND`		Query for the wind speed/direction, responds with WEATHERWIND-<speed>,direction> where <speed> will be in mph or kph, depending on scale and <direction> will be N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW

Component Functions (Cont.):					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`queryWeatherAll()`	208			`WEATHER_FORCE_READING`	Momentary Function Channel: Causes the weather station to update its readings when the channel is activated
`setWeatherScale(us)`			`WEATHERSCALE-<scale>`		Set the weather scale, where <scale> is IMPERIAL or METRIC

Listener
Name: Weather Listener
Interface: IWeatherComponentListener

Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processBarometricPressureEvent`		48		`WEATHER_BAR_LVL`	Barometric pressure changed, value is in inches Hg (Imperial Scale) or mm Hg/ torr (Metric Scale)
`processBarometricTrendEvent`	233			`WEATHER_BAR_FALLING`	Feedback Channel: Barometric pressure trend change (see chart below)
`processBarometricTrendEvent`	232			`WEATHER_BAR_RISING`	Feedback Channel: Barometric pressure trend change (see chart below)
`processChanceOfPrecipitationEvent`			`FORECASTCOP-<day1> [,<day2>,...]`		Weather forecast chance of precipitation changes, value list will contain one forecast chance of precipitation for each forecast day. The first day is always today. Values are in percent.
`processCurrentConditionEvent`			`WEATHERCONDITION-<condition>`		Weather condition changed, value will be BLIZZARD, BLOWINGSNOW, CLEAR, CLOUDY, DRIZZLE, DUST, FAIR, FOG, FREEZINGDRIZZLE, FREEZINGRAIN, HAZE, HUMID, ICE, MOSTLYCLOUDY, MOSTLYSUNNY, PARTLYCLOUDY, RAIN, RAINSHOWERS, RAINSNOWMIX, SLEET, SMOKE, SNOW, SNOWFLURRIES, SNOWSHOWERS, SUNNY, THUNDERSTORMS, UNKNOWN, VERYCOLD, WINDY
`processDewpointEvent`		47		`WEATHER_DEWPOINT_LVL`	Dewpoint changed, value is in degrees C or F depending on weather scale
`processForecastHighTemperature Event`			`FORECASTHIGH-<day1> [,<day2>,...]`		Weather forecast high temperatures changes, value list will contain one forecast high temperature for each forecast day. The first day is always today. Values are in degrees C or F depending on weather scale
`processForecastLowTemperature Event`			`FORECASTLOW-<day1> [,<day2>,...]`		Weather forecast low temperatures changes, value list will contain one forecast low temperature for each forecast day. The first day is always today. Values are in degrees C or F depending on weather scale
`processHeatIndexEvent`		46		`WEATHER_HEAT_INDEX_LVL`	Heat index temperature changed, value is in degrees C or F depending on weather scale
`processHighTemperatureEvent`		43		`WEATHER_HI_TEMP_LVL`	High temperature since midnight changed, value is in degrees C or F depending on weather scale
`processIndoorHumidityEvent`		35		`INDOOR_HUMID_LVL`	Indoor humidity changed, value is in percent
`processIndoorTemperatureEvent`		33		`INDOOR_TEMP_LVL`	Indoor temperature changed, value is in degrees C or F depending on weather scale
`processIsFreezingEvent`	231			`WEATHER_FREEZING`	Feedback Channel: Weather condition is freezing if channel is on. Weather condition is freezing when outdoor temperature is at or below 32 degrees F, 0 degrees C.

Listener Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
`processIsRainingEvent`	230			`WEATHER_RAINING`	Feedback Channel: Weather condition is raining if channel is on. Weather condition is raining when the current weather condition indicates raining.
`processLowTemperatureEvent`		44		`WEATHER_LO_TEMP_LVL`	Low temperature since midnight changed, value is in degrees C or F depending on weather scale
`processOutdoorHumidityEvent`		36		`OUTDOOR_HUMID_LVL`	Outdoor humidity changed, value is in percent
`processOutdoorTemperatureEvent`		34		`OUTDOOR_TEMP_LVL`	Outdoor temperature changed, value is in degrees C or F depending on weather scale
`processRainfallEvent`			`WEATHERRAIN-<duration>, <value>`		Rain fall changed, where <duration> is DAY, WEEK, MONTH, YEAR, YTD and <value> is in inches (Imperial Scale) or cm (Metric Scale).
`processWeatherAlertEvent`			`WEATHERALERT-<alert>`		Weather alert, where <alert> is a string containing the weather alert
`processWeatherConditionsEvent`			`FORECASTCONDITION-<day1>[,<day2>,...]`		Weather forecast conditions changed, value list will contain one forecast condition for each forecast day. The first day is always today. Values will be BLIZZARD, BLOWINGSNOW, CLEAR, CLOUDY, DRIZZLE, DUST, FAIR, FOG, FREEZINGDRIZZLE, FREEZINGRAIN, HAZE, HUMID, ICE, MOSTLYCLOUDY, MOSTLYSUNNY, PARTLYCLOUDY, RAIN, RAINSHOWERS, RAINSNOWMIX, SLEET, SMOKE, SNOW, SNOWFLURRIES, SNOWSHOWERS, SUNNY, THUNDERSTORMS, UNKNOWN, VERYCOLD, WINDY
`processWeatherScaleEvent`			`WEATHERSCALE-<scale>`		Weather scale changed, <scale> is IMPERIAL or METRIC
`processWindchillEvent`		45		`WEATHER_WIND_CHILL_LVL`	Windchill temperature changed, value is in degrees C or F depending on weather scale
`processWindInfoEvent`			`WEATHERWIND-<speed>, direction>`		Wind speed/direction had changed. Wind speed will be in mph or kph, depending on scale. Direction will be N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW

Weather Listener State Charts

processBarometricTrendEvent

State	Channel 232	Channel 233
STEADY	OFF	OFF
RISING	ON	OFF
FALLING	OFF	ON

Window

Component					
Name: Window					
Interface: IWindowComponent					
Component Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`addWindowComponent (index,windowAddress)`			`WINDOWADD-<index>, <windowAddress>`		Add a window at a given index, where <index> is 1 through x and <address> is a window address and x is the maximum supported window index (see module documentation)
`adjustBrightness(1)`	148			`BRIGHT_UP`	Momentary Function Channel: Brightness is incremented when channel is activated
`adjustBrightness(-1)`	149			`BRIGHT_DN`	Momentary Function Channel: Brightness is decremented when channel is activated
`adjustColor(1)`	150			`COLOR_UP`	Momentary Function Channel: Color is incremented when channel is activated
`adjustColor(-1)`	151			`COLOR_DN`	Momentary Function Channel: Color is decremented when channel is activated
`adjustContrast(1)`	152			`CONTRAST_UP`	Momentary Function Channel: Contrast is incremented when channel is activated
`adjustContrast(-1)`	153			`CONTRAST_DN`	Momentary Function Channel: Contrast is decremented when channel is activated
`adjustSharpness(1)`	154			`SHARP_UP`	Momentary Function Channel: Sharpness is incremented when channel is activated
`adjustSharpness(-1)`	155			`SHARP_DN`	Momentary Function Channel: Sharpness is decremented when channel is activated
`adjustTint(1)`	156			`TINT_UP`	Momentary Function Channel: Tint is incremented when channel is activated
`adjustTint(-1)`	157			`TINT_DN`	Momentary Function Channel: Tint is decremented when channel is activated
`cycleFreeze()`	213			`PIC_FREEZE`	Momentary Function Channel: Cycle freeze when channel is activated
`cyclePictureMute()`	210			`PIC_MUTE`	Momentary Function Channel: Cycle picture mute when channel is activated
`getImagePosition()`			`?IMAGE`		Query for the image position. Response with IMAGE-<source>,<x>,<y>,<height>,<width> where <source> is 1 to the maximum source number (see module documentation)
`getImageSize()`			`?IMAGE`		Query for the image size. Response with IMAGE-<source>,<x>,<y>,<height>,<width> where <source> is 1 to the maximum source number (see module documentation)
`getImageSource()`			`?IMAGE`		Query for the image source. Response with IMAGE-<source>,<x>,<y>,<height>,<width> where <source> is 1 to the maximum source number (see module documentation)
`getWindowComponentAddress (index)`			`?WINDOWADDR-<index>`		Query for the address of the window at index <index>, responds with WINDOWADDR-<index>, <address>
`getWindowComponentIndex (windowAddress)`			`?WINDOWIDX-<address>`		Query for the index of the window with address <address>, responds with WINDOWADDR- <index>, <address>
`getWindowPosition()`			`?WINDOW`		Query for the window position. Response with WINDOW-<x>,<y>,<height>,<width>
`getWindowSize()`			`?WINDOW`		Query for the window size. Response with WINDOW-<x>,<y>,<height>,<width>
`getZOrder()`			`?WINDOWZORDER`		Get the Z-order. Responds with WINDOWZORDER-<value> where <value> is 1 to the maximum z-order of the device, 1 is the top-most z-order (see module documentation)
`pan(DOWN)`	133			`PAN_DN`	Momentary Function Channel: Image is moved down one step within the window when channel is activated

Component Functions (Cont.):

Name:	Channel:	Level:	Command:	Constant:	Notes:
`pan(LEFT)`	134			`PAN_LT`	Momentary Function Channel: Image is moved left one step within the window when channel is activated
`pan(RIGHT)`	135			`PAN_RT`	Momentary Function Channel: Image is moved right one step within the window when channel is activated
`pan(UP)`	132			`PAN_UP`	Momentary Function Channel: Image is moved up one step within the window when channel is activated
`removeWindowComponent (index)`			`WINDOWREMOVEIDX-<index>`		Remove the window at index <index>, where <index> is 1 through x and x is the maximum supported window index (see specific module documentation)
`removeWindowComponent (windowAddress)`			`WINDOWREMOVEADDR-<windowAddress>`		Remove the window with address <address>, where <address> is a window address
`setBrightness(level)`		10		`BRIGHT_LVL`	Set brightness level, range is 0-255
`setColor(level)`		11		`COLOR_LVL`	Set color level, range is 0-255
`setContrast(level)`		12		`CONTRAST_LVL`	Set contrast level, range is 0-255
`setFreezeOn(state)`	214			`PIC_FREEZE_ON`	Discrete Function Channel: Freeze is on while channel is active
`setImage(source,pos,size)`			`IMAGE-<source>,<x>,<y>,<height>,<width>`		Set the image source, position and size relative to the window where <source> is 1 to the maximum source number (see specific module documentation) and <x> and <y> are the window coordinates for the upper-left hand corner of the image and <height> and <width> are the size of the image to be displayed in the window. Used to change the position of the image visible in the window.
`setImageSource(int)`			`IMAGESOURCE-<source>`		Set the image source for the window where <source> is 1 to the maximum source number (see specific module documentation)
`setPictureMuteOn(state)`	211			`PIC_MUTE_ON`	Discrete Function Channel: Picture Mute is on while channel is active
`setSharpness(level)`		13		`SHARP_LVL`	Set sharpness level, range is 0-255
`setTint(level)`		14		`TINT_LVL`	Set tint level, range is 0-255
`setWindow(pos,size)`			`WINDOW-<x>,<y>,<height>,<width>`		Set the window position and size relative to the display where <x> and <y> are the display coordinates for the upper-left hand corner of the window and <height> and <width> are the size of the window to be displayed in the display. Used to change the position of the window visible in the display.
`setZOrder(position)`			`WINDOWZORDER-<position>`		Shuffle the window z-order relative to the other windows where <position> is FRONT, BACK, FORWARD (up one level), BACKWARD (down one level)
`setZOrder(value)`			`WINDOWZORDER-<value>`		Set the Z-order where <value> is 1 to the maximum z-order of the device, 1 is the top-most z-order (see specific module documentation)
`zoomIn()`	159			`ZOOM_IN`	Momentary Function Channel: Image size is enlarged within the window (zoom in/tele) when channel is activated
`zoomOut()`	158			`ZOOM_OUT`	Momentary Function Channel: Image size is reduced within the window (zoom out/window) when channel is activated

Listener					
Name: Window Listener					
Interface: IWindowComponentListener					
Listener Functions:					
Name:	**Channel:**	**Level:**	**Command:**	**Constant:**	**Notes:**
`processBrightnessEvent`		10		`BRIGHT_LVL`	Brightness changed, range is 0-255
`processColorEvent`		11		`COLOR_LVL`	Color changed, range is 0-255
`processContrastEvent`		12		`CONTRAST_LVL`	Contrast changed, range is 0-255
`processFreezeEvent`	214			`PIC_FREEZE_FB`	Feedback Channel: Freeze is on if channel is on
`processImageEvent`			`IMAGE-<source>,<x>,<y>,<height>,<width>`		Image position and/or size changed where <source> is 1 to the maximum source number (see specific module documentation)
`processImageSourceEvent`			`IMAGESOURCE-<source>`		Image source for the window changed where <source> is 1 to the maximum source number (see specific module documentation)
`processPictureMuteEvent`	211			`PIC_MUTE_FB`	Feedback Channel: Picture is muted if channel is on
`processSharpnessEvent`		13		`SHARP_LVL`	Sharpness changed, range is 0-255
`processTintEvent`		14		`TINT_LVL`	Tint changed, range is 0-255
`processWindowEvent`			`WINDOW-<x>,<y>,<height>,<width>`		Window position and/or size changed.
`processZOrderEvent`			`WINDOWZORDER-<value>`		Z-order changed where <value> is 1 to the maximum z-order of the device, 1 is the top-most z-order (see module documentation)

Last Revised:
11/17/2015