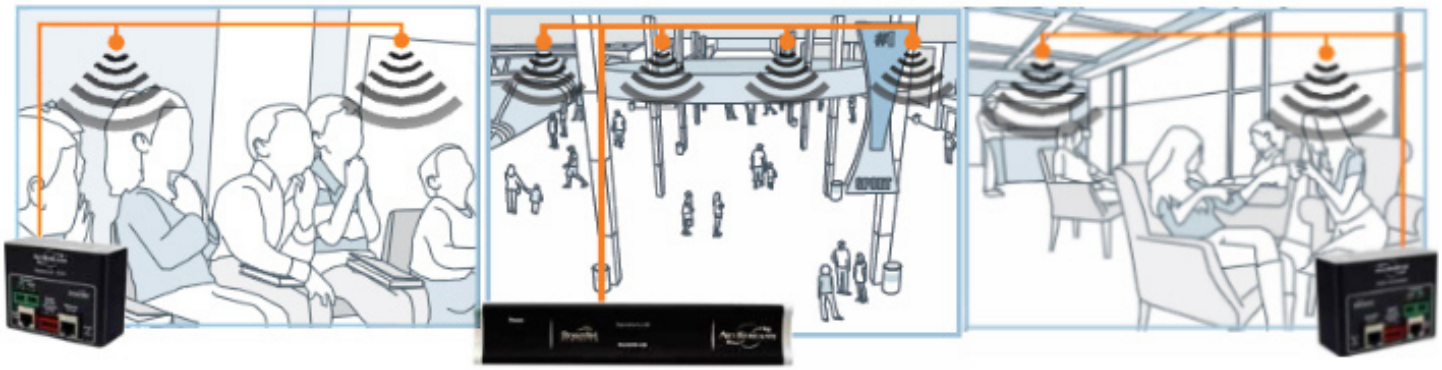

StreamNet®

Integration & Programmer's Guide

Telephone	1.800.283.5936 1.801.974.3760
Tech Sales	1.800.705.2103
FAX	1.801.974.3669
E-mail	tech.support@clearone.com techsales@clearone.com support@netstreams.com
On the Web	www.clearone.com www.netstreams.com www.streamnetpartners.com

StreamNet

Integration & Programmer's Guide

CLEARONE PART NO. 800-000-017 APRIL 16, 2011 (REV. 1.0)

U.S. PATENTS: 7,643,894 & 7,711,126

INTERNATIONAL PATENTS: 2003-241405 (AU), 10-0966415, 10-2004-7017970 (KO)

OTHER PATENTS PENDING:

US 11/467,340
US 11/960,401
US 12/015,385
US 12/418,267
US 12/472,976
US 12/727,925
US 12/761,506
US 12/900,666
US 12/917,765
US 12/917,773
European 03731139.6
European 06251345.2
Canadian 2,485,104
Canadian 2,539,458
Australia 2008-207498

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Chapter 1: Introduction

DOCUMENT PURPOSE

The intent of the StreamNet Integration & Programmer's Guide is to define the StreamNet software command structures necessary for Integrators to build interfaces. The interfaces can then be used by StreamNet System Controllers to control StreamNet Ready devices.

DOCUMENT ORGANIZATION

The StreamNet Integration & Programmer's Guide is designed to provide the Integrator with detailed descriptions of the relevent commands for each of the StreamNet device sets (SpeakerLinX, MediaLinX, TouchLinX, ViewLinX, etc.) Each set of commands is organized based on the specific sources available and are listed through-out in alphabetical order.

This document begins with a detailed overview of the StreamNet ASCII Command Protocol which is the basic mechanism for control and status reporting of the StreamNet services.

Chapter 2 provides detailed information needed for controlling a StreamNet System from connection to addressing. Information on controlling legacy AV devices, StreamNet Soft Adapters (SSA), Source Adapters as well as Display and Control Adapters is also covered. The chapter also covers Streamnet Subscription and Registration topics.

Chapter 3 gives an overview of StreamNet ASCII commands, with an explanation of syntax structure, Subnode addressing and related information. Chapter 3 provides details on ASCII message types and provides examples as necessary. The chapter finishes with sections on Menu Navigation, Report Messages, ASCII Character Escape Sequences and Streamnet macros.

Chapter 4 provides the reader with details of the services which may be found on a StreamNet device. Sections include a brief description on communicating with the StreamNet system, a description of what a service is and the types of services available. A chart of common services by device is provided and some examples of zone configurations are presented.

Chapter 5 describes how services are implemented and examples of commands available per service type with details of the parameters available for each listed command. Each service section of Chapter 5 provides a detailed overview of the service type.

Chapter 5 is followed by an Index that can be useful as a reference source.

The document concludes with a Glossary of Terms.

Chapter 2: Controlling A StreamNet System

StreamNet is designed such that a very large number of networked devices behave as a single coordinated system. The integrator does not need to know about the underlying network complexity; StreamNet completely handles it all. With StreamNet you get all of the advantages of an IP based system without the complexity. The third party control device only needs to know a single IP address and through this one address can control an unlimited number of StreamNet devices. Instead of IP addresses (which may change) the controller can address commands to friendly, human readable names such as "Bedroom TV" or "DVD Player". Control messages using these friendly names may be sent to any StreamNet device or broadcast to all of them; StreamNet will make sure that the correct device(s) and only the correct device(s) receive the message.

- **NOTE:** This document will describe a large number of alternative ways to control a StreamNet system. This is a testament to the incredible flexibility of StreamNet; but, it may give the impression that controlling StreamNet is complex – nothing could be further from the truth. The integrator does not need to use all of the methods, only the most convenient one(s).

A StreamNet system remains remarkably uncomplicated even when scaled to huge proportions. Network technology makes this scalability possible and StreamNet makes it simple.

Touchpanel user interface devices are available for controlling StreamNet without requiring any central controller; but, this by no means the only way a StreamNet system can be controlled. In fact, StreamNet was explicitly designed to simplify control by third party control devices such as AMX or Crestron Controllers.

Advanced users may be interested in StreamNet's Auto-Discovery or Time Sync mechanisms, but the integrator may mostly ignore how StreamNet does what it does and just take advantage of the simplicity it offers.

StreamNet devices handle the routing of AV content from one location to another as well as all aspects of controlling the source and display devices. Most devices can learn and reproduce IR codes for controlling non-StreamNet devices such as TVs, DVD players, etc. Serial and IP control of non-StreamNet devices are also supported. StreamNet can even route IR signals transparently across the network to be reproduced at other locations to control remote devices.

CONNECTING TO THE STREAMNET SYSTEM

Control messages may be sent to the StreamNet system in a variety of ways. Messages may be sent through a TCP connection to a single device or through multiple TCP connections. Also, messages may be sent using UDP, either unicast or multicast. It is even possible to send the commands using an IR or RS232 serial interface. The integrator may choose the most convenient method or any combination of these methods.

StreamNet TCP Portal

The most common way for a controller to connect to the StreamNet system is through a single TCP connection to one of the StreamNet devices. This single TCP connection can be made to any StreamNet device which supports the "TCP Control Portal". If lots of control message traffic is expected then it is recommended that a ControLinX be dedicated to providing this TCP connection. The commands are sent and responses and unsolicited status messages are conveyed over the TCP connection. The TCP connection should be maintained continually if system status is desired. Typically, each StreamNet device supports a maximum of 4 TCP clients at a time.

Obviously to make the TCP connection the controller must know the IP address of the device to which it is connecting. So at least one device in the StreamNet system must have a fixed IP address. This is another good reason to dedicate a ControLinX device as the portal to the StreamNet system. The portal ControLinX can be assigned a static IP address but all of the other StreamNet devices can be allowed to find their own address.

- **NOTE:** If there is no traffic on the TCP connection after 60 seconds it will be closed. To keep the connection alive the client can send a dummy ASCII command every 30 seconds. #HEARTBEAT is often used for this purpose. But if the client has registered for status the #REGISTER commands will keep the connection alive.

TCP Control

Any StreamNet ASCII control message may be sent using TCP to the IP address of any StreamNet device at port number 15000. Responses from the StreamNet system will be returned on the TCP connection which initiated the request. See chapter on ASCII Protocol.

TCP Monitoring

Unsolicited status messages may be received by the third party controller using TCP through the StreamNet mechanism of Registration. See section on Registration.

StreamNet UDP Portal

A third party controller may perform all StreamNet control and monitoring functions using UDP.

Control Messages may be sent to the StreamNet system either multicast or unicast; monitoring or status messages may be received either multicast or unicast. Any or all of these methods may be used simultaneously in any desired combination.

Unicast UDP Control

Any StreamNet ASCII control message may be sent using UDP to the IP address of any StreamNet device at port number 15000. Responses from the StreamNet system will be addressed and routed according to the ASCII "From Address" of the original message. See chapter on ASCII Protocol.

Multicast UDP Control

Any StreamNet ASCII control message may be sent using UDP to the StreamNet System Multicast Address at port number 15000. Responses from the StreamNet system will be addressed and routed according to the ASCII "From Address" of the original message. See chapter on ASCII Protocol.

Unicast UDP Monitoring

Unsolicited status messages may be received by the third party controller using unicast UDP through either of two StreamNet mechanisms, Subscription or Registration.

Multicast UDP Monitoring

Unsolicited status messages may be received by the third party controller using unicast UDP through the StreamNet mechanism of Subscription. It is not recommended for the controller to monitor the multicast status of the services directly. This mechanism is not intended for this use and may change without notice.

StreamNet Compatibles

Some devices support only a subset of the StreamNet protocols and functionality; these devices are sometimes referred to as StreamNet Compatible devices. These devices currently include the iDock and the Anthology Media Server. These devices work seamlessly within a StreamNet system but do not provide all of the StreamNet functionality described in this document.

STREAMNET CONCEPTS

There are few concepts that must be understood to properly understand StreamNet.

Services

A StreamNet service is an abstraction. It is a functional software module but it can be thought of as a virtual device. A service performs a single function such as capturing analog audio and streaming that audio onto the network. Another service may perform the function of converting the network audio stream to analog and amplifying it. Still another service may provide an interface to a third party lighting system. A single physical device may support many services of various types.

The service concept greatly simplifies controlling the StreamNet system. For example, control messages can be addressed to the name of an AV zone (a service) instead of to "Device 1 at IP address, 10.15.65.113, output number 7". The controller does not need to know which physical device is providing the function or what the physical configuration of that device is – just the service name. The control messages are simply addressed to "Zone 1" and StreamNet ensures the correct AV zone reacts. Details about the types of services and the functions performed are described in later sections of this document.

STREAMNET ADDRESSING

StreamNet ASCII messages may be sent, using any of various unicast methods, which include UDP (Unicast) and TCP, to any StreamNet device. If addressed correctly, the message will be routed to the desired service(s). Also, the messages may be sent to the StreamNet System multicast address. In this case, all StreamNet devices receive the message simultaneously, but only the addressed services accept the message and react to it. Messages may be addressed to a Service, a Group or a Room.

It is not necessary to know the IP addresses of the individual StreamNet devices to communicate with them. All messages are addressed to service names associated with the devices during the configuration process.

If the 3rd Party Controller or software has the ability to send Multicast UDP messages, then all ASCII messages may be sent to the default system Multicast Address and the destination device or group of destination devices will receive it. Responses and status may be received by unicast/multicast UDP. If the 3rd Party Controller or software does not have multicast capability, the messages may be sent Unicast UDP to any StreamNet device and it will be forwarded.

If the 3rd Party Controller is also listening to its own (i.e., different) multicast address for responses, the controller and the StreamNet system do not have to be in the same subnet.

The StreamNet system multicast address should not be used for responses. 3rd Party Controllers should subscribe to use a separate unicast address or an unused multicast address to receive responses (see the section on ***Subscription*** below).

Unsolicited status messages may be received by the 3rd Party Controller.

Group Addresses

Services can be members of named "groups". A group address, or "group name", is just a way of addressing multiple services simultaneously with a single control message. A service may be a permanent member of a particular group or it may join and/or leave the group. There may be special group names which are used for specific purposes; for example, the group name "ALL" includes all services in the StreamNet system.

Controlling Legacy AV Devices

Driver Pass Through (Serial or IP)

Generally, when controlling an AV source device such as a media server or Sirius tuner the integrator will want to use the drivers which already exist for their third party controller. StreamNet makes this simple and convenient. The serial ports on any StreamNet device can be configured to transparently pass serial traffic to/from any controller. In this way the existing driver can be used and no additional hardware is required to add a serial port. And of course if the driver uses Internet Protocol (IP) to control the device this can be done just as easily.

StreamNet Soft Adapters (SSA)

Another StreamNet feature which greatly simplifies control is the StreamNet Soft Adapter. Adapters are small pieces of software written in the LUA programming language and executed on devices throughout the StreamNet system. Soft adapters convert proprietary protocols of third party devices into standard StreamNet protocol. This eliminates the complexity caused by multiple protocols from multiple vendors. Since the Soft Adapters are executed by an "interpreter", different Adapters may be selected by the integrator during system configuration from a large number of standard adapters or the integrator may write a custom adapter to perform any desired function.

Source Adapters (Drivers)

Source Adapters are software translators (drivers) which standardize the control of third party AV source devices. Source Adapters take the myriad of device control interfaces and translate them to the common language of StreamNet.

For example, media servers from different vendors may have very different and complex proprietary interface protocols; but, Source Adapters can make them all operate exactly the same from the controller's point of view. Source Adapters may use IR, Serial, of Internet Protocol (IP) to control the AV source device.

The integrator may choose from a number of standard Source Adapters or the integrator may elect to write a custom driver. If the integrator prefers to take advantage of existing drivers on third party controllers, StreamNet devices can transparently pass serial communications to/from the AV device or learned IR codes can be triggered by command.

Source Adapters run on devices with AV source services or Stream Proxy Services, such as a MediaLinX or SpeakerLinX.

Display Adapters (Drivers)

Some StreamNet devices also support Display Adapters to translate controls for the video display device.

Control Adapters (GPIO Drivers)

Some Adapters can provide an interface to lighting systems or HVAC systems, or the Adapter may have multiple unrelated functions. A custom Soft Adapter may perform multiple functions simultaneously. These types of Adapters are called Control Adapters or GPIO drivers. GPIO Adapters can run on any StreamNet device that has a GPIO service which supports Adapters; a ControLinx is a typical example. Control Adapters can be custom written to perform nearly any conceivable control task.

The control messages themselves will be described in later chapters as the StreamNet ASCII Protocol.

> ***Adapter (computing) – Wikipedia***
>
> ***"In computing, an adapter is a hardware device or software component that converts transmitted data from one presentation form to another. The data presentation can be, for example, a message sent between objects in an application or a packet sent through a network."***

SUBSCRIPTION

Receiving System Status

StreamNet Subscription allows third party controllers to receive unsolicited status messages from StreamNet services. During configuration, the Integrator may subscribe to a StreamNet Service's status message by entering the controller's IP address and the port to which the status messages are to be sent.

If a subscription exists, the StreamNet service will send status messages over UDP protocol to the specified IP address and port number.

A static subscription is configured using the ***StreamNet Dealer Setup Program*** and is automatically in effect as soon as the device powers up and remains in effect indefinitely. The user should use the ***StreamNet Dealer Setup Program*** to specify an IP address and port number to which status reports are to be sent. This IP Address is typically the static IP address of the 3rd Party Controller device. If the device receiving the status messages is capable of receiving multicast UDP messages, then an unused multicast address may be specified for the subscription. Using a multicast address eliminates the need for the controller to have a static IP Address and also eliminates any subnet related issues.

During configuration, the integrator may enter "subscriptions" to a StreamNet service's status messages by entering the IP address and port to where the status messages are to be sent. If a subscription exists the StreamNet service will send status messages over UDP protocol to the specified IP address and port number.

The StreamNet system multicast address should not be used by the controller to receive responses or status. When Controllers "Subscribe" either the controller's unicast address or an unused multicast address should be used.

REGISTRATION

When controlling the system through a TCP connection the TCP client may "register" to receive status from individual services. Registration is performed by sending a "**#REGISTER**" ASCII command to the StreamNet device acting as the TCP server. This registration will time out after 30 seconds unless it is renewed by sending another "**#REGISTER**" command.

A TCP client of a StreamNet device may "register" to receive status from a specific service. When a TCP client such as a Flash application sends the "**#REGISTER** serviceName" command to the root service of the device to which it is connected that device will start forwarding the status of the specified service to the TCP client. This function is useful for controllers which cannot use multicast UDP. Adobe Flash can only make TCP connections.

STREAMNET MULTICAST ADDRESSES

For each StreamNet system there is a single multicast address used by all of the devices for device discovery, ASCII message routing, time synchronization, and other system level tasks; this multicast address is called the "StreamNet System Multicast Address" and is configured into the device by the StreamNet set up program. All other multicast addresses are assigned dynamically at run time by the StreamNet system master device. The assigned addresses are chosen from a range of addresses configured into the device by the StreamNet set up program.

These dynamically assigned multicast addresses would not normally be needed by any third party controllers or other non-StreamNet devices; however, the addresses may sometimes be available through **#QUERY** requests or other status reports.

IP ADDRESS ASSIGNMENT

Each StreamNet device must have an IP address. The device may obtain this address in any of several ways: Static IP, Auto IP or DHCP. The installer may choose a preferred method of IP address assignment but if the preferred method fails, the device may try other methods until an address is obtained. By default the devices will initially try to obtain an address from a DHCP server. If after several minutes an address is not received from a DHCP server, the device will attempt to obtain an address through the Auto-IP method. With the Static IP method, the device will assume that the assigned address is valid.

- **NOTE:** If the device is configured for DHCP and no DHCP server is present on the system, the device will remain unresponsive for several minutes until the Auto IP address is resolved.

IP Port Number Table	
Multicast UDP ASCII Messaging	Port 15000
Unicast UDP ASCII Messaging	Port 15000
TCP ASCII Messaging	Port 15000
Discovery	Port 8000
Time Sync	Port 5001

Chapter 3: ASCII Overview

The basic mechanism for control and status reporting of the services in StreamNet is the ASCII protocol. This protocol consists of various command messages and their parameters represented by strings of ASCII characters. Messages also have associated routing and administrative information which informs recipients of the destination and origin of the messages or provides other supplemental information.

ASCII MESSAGE SYNTAX

ASCII messages have the following syntax:

#@toAddress:fromAddress%modifier%modifier#keyword arg1, arg2, ... argN /0

- A # symbol is required at the beginning of all messages.
- ASCII messages must not exceed 1000 characters in length.
- The various fields must be in the order shown above.
- The fields after the first # are the administrative fields, addresses and other control functions. The fields after the second # are the actual command.

@toAddress

Specifies the name of the Service, Group, or Room to which the message is sent. The **@** symbol precedes all **toAddress** fields. The address field may consist of two parts: the **Node Address** and the **Subnode Address**. The Node Address specifies the Service, Group, or Room the message is addressed to. The Subnode Address can be used to address a subordinate module within the destination service or to invoke other special routing functions. The Subnode Address follows the Node Address and is separated by a tilde (~).

Example: **@nodeAddress~subnodeAddress**

- **NOTE**: When no service address is provided, the address defaults to the Root Service of that device.

  The device reacts only if the command is a valid Root Service command.

  Messages without a toAddress will be ignored if they are sent multicast or broadcast.

Node

A Node Address is the main part of an ASCII command address and can be part of the TO or FROM address. A node address is either a Service Name, a Service Unique ID (UID), a Room Name or a Group Name.

Subnode

A subnode address is part of the ASCII command address. It can be part of the TO or FROM address. A subnode address is a special address modifier which instructs the node where to forward the command once it is received. Sometimes the subnode address indicates that the service should forward the command to a non-StreamNet device.

Examples:

- **~IRMOD**: forwards the command to the device which is providing the IR I/O functions to the StreamNet service.
- **~UDP192.168.5.3_5000**: forwards the command using UDP to the specified IP address and port 5000. The subnode address may refer to a StreamNet device with an address unknown to the sender.

StreamNet Subnode Addresses:

- **~CURSRC** Route to renderers current audio source
- **~IRMOD** Route to IR module on MediaLinX
- **~SERIAL_X** Route to serial port number X
- **~KEYPAD** Route to KeyLinX module
- **~ROOT** Route to root service on device
- **~STATUS** Not used to route to, but rather to indicate a status message
- **~SUBSCRIBER** indicates the route to subscribers

:fromAddress

:fromAddress indicates which entity sent the message. This is used by the recipient to address any response that may be generated by the message. If a response is expected, then the fromAddress should specify the address the response should be sent to. The **:** symbol must precede all fromAddress fields. The fromAddress may have a subnode address.

For 3rd Party Controllers using UDP, it is recommended that the fromAddress resemble the following format:

~UDP192.168.5.3_5000

This includes the IP address and port to which any response should be sent.

%modifier

Optional field. May be used for various context specific purposes. If the recipient does not support the particular modifier, the field may be ignored. There may be any number of **%modifier** fields in a message. The **%** symbol precedes all modifier fields.

- **NOTE**: This field is reserved for future use in StreamNet.

#keyword (ASCII Command)

Required field. This is the keyword that specifies the action to be executed. The **#** symbol precedes all keyword fields (no intervening spaces). The keyword is one word and contains no spaces, commas, or other special characters except the underscore "_".

arg1, arg2, ..., arg#

Optional field (some keywords may require arguments). The keyword may be followed by arguments. There are one or more spaces between the keyword and the first argument and a comma between each additional argument. Arguments are separated by commas and any extra white space should be ignored.

Arguments have the following formats:

- **Meaningful Strings** – These are labels which represent some value, case, or function. Examples of meaningful strings are ON and OFF. Meaningful strings are not enclosed in quotation marks and must begin with an alphabetic character – not a numeral.
- **Character Literals** – These character strings do not need any special delimiters if they contain no blanks, spaces, commas, or other special characters. (A special character is other than alpha/numeric).

▶ **NOTE**: If an argument contains any special characters or spaces, it must be enclosed between double braces **{{.....}}**. When in doubt, use the double braces.

- **Numeric Scalars** – These must be a decimal number.
- **XML or other Special Formats** - These arguments are enclosed in double braces **{{...}}** and can take any form depending on context.

/0 (NULL BYTE)
All ASCII messages must end with a 0x00 byte (null).

UniqueID (UID)

The UID of a service is an unchangeable identifier for the service. It is based upon the device's serial number. The UID will be the service name if none has been specified during configuration. It will also be used as the service's room name if none has been specified. Normally, there is no reason for an integrator to use the UID. All services should be assigned names and all of the user's commands should be addressed to those service names (or room/group names).

The structure of the UID is serialNumber_index where **serialNumber** is the Serial Number of the device and **index** is an integer that identifies the specific Service on the device.

Example: **#@MLA10105151001161008019_4**

serialNumber **index**

Summary

The 3rd Party Controller sends an ASCII message UDP to the IP address of any StreamNet device.

In the ASCII message, you should have the:

- **#@toAddress** - name of the service you are addressing.
- **:fromAddress** - your own IP Address with a Port number to receive responses.
- The **:fromAddress** - your IP Address/Port and all responses will be sent to this address.
- **#ASCII** command with appropriate arguments required by that command.

ASCII Message Types

There are three types of ASCII messages - **Imperative**, **Interrogative** and **Declarative**.

Imperatives

Some messages are commands which initiate an action or directly cause a change of state. For example, commands that change volume level, select an audio source, or send an IR code.

Example: **#@CD PLAYER:~UDP192.168.5.3_5000#PLAY**

The message will be routed to the StreamNet Service named CD PLAYER. This service may transmit an IR code for PLAY to the actual CD player being controlled.

- **NOTE**: This command could be received by any StreamNet device in the system and it would be routed to the correct device.

Generally, imperative StreamNet commands do not receive responses.

Interrogatives

Some messages are used to request information. If an interrogative message is sent, a response is expected.

- **NOTE**: An Imperative may cause a change of state which then generates a report of the change. This is different from the question/answer of the Interrogatives.

Declaratives

Some messages only report information. They may be unsolicited status reports (periodic or caused by a change of state) or they may be a response to an interrogative message. The most common StreamNet declarative is the **#REPORT** message.

Menus

StreamNet has a special set of messages which make use of all three types, interrogative, declarative, and imperative. These messages are the menu messages: **#MENU_LIST**, **#MENU_RESP**, and **#MENU_SEL** These three messages combine to provide a general means to present menus and allow menu selections without knowing the content of the menu in advance.

The controller sends a **#MENU_LIST** message to a StreamNet service to request the menu. The StreamNet service responds with a series of **#MENU_RESP** messages (one for each item in the menu). After a selection has been made, the controller sends a **#MENU_SEL** message indicating the user's selection. In addition to user readable menus selection mechanism, the **#MENU_LIST** can be used to provide information to third party control programs.

Menus are organized in a branching tree structure. At the top level of the menu will be a list of items. Selecting one of these items may lead to a second tier menu; selecting a second tier item may lead to a third tier menu, etc. This may continue for an arbitrary number of menu tiers. At any level of the menu, there may be some of the items which do NOT lead to a lower tier. These are called "terminal" items and generally selecting (use **#MENU_SEL**) a terminal item will initiate some action in the system.

The syntax of the three menu messages are:

#MENU_LIST StartingItemNumber, EndingItemNumber, {{MenuPath}}

Where:

StartingItemNumber is the item number of the first item to be returned in the menu response

EndingItemNumber is the last item to be returned

{{MenuPath}} is the navigation path for the current menu tier

#MENU_RESP {{<MenuName id="itemId" children="0" itemnum="1" idpath="tier1>tier2>etc" disppath="TierName" display="Item ID" otherData="123456" moreData="xyz" />}}

The StreamNet service will respond to a **#MENULIST** message with a series of menu response messages, one for each menu item requested. The payload of the menu response message is XML-like, items may or may not be present and the order of the listed items may change.

The following are some of the items that will always be present:

id	This is the identifier for this item.
display	This is the text shown in the menu for this item. This should be a more readable version of the "id".
children	This is how many items are in the next tier menu below this item. 0 indicates that this is a terminal item, and there is no menu tier below this item.
itemnum	This indicates the item number within the menu list. Receiving itemnum="-n" (any negative integer) indicates that there are no more items at this tier of the menu.
idpath	This is the menu navigation path taken to arrive at this menu tier. The path is specified by a list of "id" fields separated by the greater-than sign, ">".
disppath	This is a more readable version of the idpath to be used on user interfaces.
MenuName	Generally, this may be ignored.

In addition to the required items there may be any number of context specific items. These items provide additional information but are only defined for that specific menu.

#MENU_SEL {{MenuPath}}

Where **{{MenuPath}}** is the navigation path of the item being selected. The MenuPath is created by concatenating the **idpath** with the **itemId** as in: **{{idpath>itemId}}**.

- **NOTE:** Do not use the **#MENU_SEL** outside of the context of menu navigation. Menus can and will change.

Menu Navigation

The menu navigation process begins with the controller or user interface device requesting the top level of a menu. Menus can be very long so it is recommended that requests specify a small range of menu items to be returned. The controller must then repeat the **#MENU_LIST** command changing the specified range until it has all of the menu items desired.

The requestor must request the menu by name:

#MENU_LIST 1,6, {{MenuName}}

In response the requested items are returned, one **#MENU_RESP** for each item requested plus one if the end of the list is reached.

#MENU_RESP {{<MenuName id="itemId" children="0" itemnum="1" idpath="MenuName" disppath="Menu Name" display="Item ID" otherData="123456" moreData="xyz" />}}

Menus can be very long so it is recommended that requests specify a small range of menu items to be returned. The controller must then repeat the **#MENU_LIST** command changing the specified range until it has all of the menu items desired. When all of the items in the menu tier have been sent a separate **#MENU_RESP** message will be sent with itemnum="-n".

- Note that if exactly the maximum number of items are requested then an additional and final **#MENU_RESP** will be returned with itemnum="-n".

The user interface should display a list of the items' "display" fields.

Next the user selects an item from the menu.

If the item is a terminal item the UI device sends a **#MENU_SEL** command specifying the "path" of the item selected to indicate that an action has been requested.

#MENU_SEL {{MenuName>tier1ItemId}}

If the item is not terminal then the UI device will send a **#MENU_LIST** message to request the next tier of the menu:

#MENU_LIST StartingItemNumber, EndingItemNumber, {{MenuName>tier1ItemId}}

The StreamNet service then returns the next tier of the menu. There can be an arbitrary number of menu tiers.

Path names can be of arbitrary length. Some example path names are:

{{ media}}

{{ media>rock}}

{{ media>rock>artists}}

{{ media>rock>artists>beatles}}

{{ media>rock>artists>beatles>white album}}

{{ media>rock>artists>beatles>white album>helter skelter}}

{{sources>dvd1}}

#REPORT MESSAGES

The primary mechanism for a StreamNet service to convey its current state and status is by sending **#REPORT** ASCII messages. **#REPORT** messages may be sent in response to a **#QUERY** command or they may be sent unsolicited.

A **#REPORT** sent in response to a **#QUERY** will be routed as an ordinary ASCII message. However, an unsolicited **#REPORT** will not be routed as an ASCII command.

The "FromAddress" of the **#REPORT** message will indicate which service sent the **#REPORT**. Usually, the FromAddress will have the subnode address, "~STATUS" attached to it.

The payload of a **#REPORT** message is XML-like, populated with label/value pairs of the form, **<LABEL="value">**.

#@ToAddress:FromAddress~STATUS#REPORT {{<reportType LABEL1="VALUE" LABEL2="VALUE" LABEL3="VALUE"/>}}

The **reportType** field should not be used for any purpose. Only the FromAddress and the label/value pairs are significant. The label/value pairs may be in any order. A particular **#REPORT** may omit some label/value pairs or new label/value pairs may be added at any time. Any unrecognized label/value pairs should be ignored.

Some special characters in the **#REPORT** message will be "escaped", using HTML style escape sequences. For example, the VALUE field cannot contain a quotation mark, so every quotation mark that is part of the VALUE field will be replaced with " or "

Example:

<ReportTypeTitle="HeSaid,"YES!""> or **<ReportTypeTitle="HeSaid,"YES!"">** (See **Character Escape Sequences** section for more information).

Label/value pairs will differ from device type to device type or service type to service type (See the documentation for each device/service for specific label/value pairs supported.

#QUERY/#REPORT Messages

Upon receiving a **#QUERY** message, a StreamNet service will send a **#REPORT** message with the requested information. The solicited **#REPORT** message will addressed to the "**FromAddress**" of #**QUERY** message. The the response may be split into multiple **#REPORT** messages. No message may exceed the 1000 character limit.

#QUERY/#REPORT Example

The "**Touchpanel**" service sends:

#@Kitchen Player:Touchpanel#QUERY current_source

The "**Kitchen Player**" service responds with:

#@Touchpanel:Kitchen Player#REPORT {{<report type="state" currentSource="myXM" currentSourceIP="10.15.96.149" permId="MLA10105151001161008019_1" />}}

Unsolicited #REPORT

Unsolicited **#REPORT** messages are sent periodically and when the status of the service changes. The unsolicited **#REPORT** messages are sent through any of several mechanisms.

Most commonly the unsolicited **#REPORT** message is sent using UDP on the multicast IP address which has been assigned to the service for that purpose. This multicast address is referred to as the service's "**Status Multicast Address**" (See **Multicast Addresses** section).

Another common method of receiving unsolicited **#REPORT** messages is through "registration". A TCP client may register to receive unsolicited **#REPORT** messages by sending a **#REGISTER** message to the root service of the device to which it is connected (See **TCP Client** section).

During device configuration a UDP address and port number may be designated to receive all unsolicited **#REPORT** messages. This is called "subscribing to the status of a service" or "subscription".

For a central controller type system topology, "subscription" is the recommended method of receiving status.

ASCII ADDRESSING AND ROUTING FOR TCP CONTROL

A common way for third parties to control a StreamNet system is through a TCP connection. The Controller device makes a TCP connection (Port 15000) to any StreamNet device. The third party controller is the TCP client and the StreamNet device acts as the TCP server. Typically each StreamNet device supports a maximum of 4 TCP clients.

If a whole system is to be controlled through this connection, it is recommended that a dedicated StreamNet device such as a ControLinX be used for this purpose. Once a TCP connection is made the controller may send commands and receive responses to **#QUERY** messages. Any messages sent by the controller should use the proper "ToAddress" but should completely leave off the "FromAddress". The StreamNet router will add the appropriate "FromAddress" which will be of the form "rootServiceName~TCPn.n.n.n_pppp", where n.n.n.n is the IP address of the TCP client and pppp is the port number of the TCP connection.

An example of how to address a command sent through the TCP portal:

#@ToWhom#COMMAND

To receive unsolicited status reports the device must send #REGISTER messages to the root service of the device to which it is connected. A **#REGISTER** message is required for every service that the controller wishes to monitor. The **#REGISTER** command times out after 30 seconds, so the **#REGISTER** commands must be repeated for as long as the controller desires to receive status reports. The repeating **#REGISTER** commands can be used as a heart beat to keep the TCP connection open.

An example of a typical **#REGISTER** command:

#REGISTER {{ServiceName}}

Notice that there is no ToAddress and no FromAddress. Leaving off the ToAddress causes the message to be sent to the local root service and the FromAddress will be filled in by the StreamNet routing function.

ASCII CHARACTER ESCAPE SEQUENCES

Sometimes characters which have a special meaning in an ASCII-based protocol will occur within text strings; therefore, there must be a way to indicate that the character in a particular instance does NOT convey its special meaning. This situation arises for StreamNet menu responses and status reports which have an XML style format. The double quotation mark cannot occur within a string of characters without being confused with the double quotation marks meant to enclose the string.

World Wide Web protocols use several methods for escaping characters and unfortunately all of these methods have been used by StreamNet drivers in the past. Henceforth, the StreamNet ASCII protocol shall use only the HTML style "&#nnnn;" format for escaping characters within literal strings (nnnn represents a decimal number of multiple digits). There should not be a need to escape any characters besides the double quotation mark but parsers should be prepared for any character to be escaped.

For historical purposes the various methods previously used for escaping the double quotation mark (") are listed here.

The double quotation mark, ", has been represented variously as:

"

"

"

%22

\"

STREAMNET MACROS

During configuration of the StreamNet system the integrator may define a quantity of "macros". Macros are a list of StreamNet commands that when invoked get executed in order. Variable time delays maybe added between the execution of the commands. It is usually a StreamNet Root service which executes a macro.

When a macro is defined the integrator must give the macro a unique name. This name is later used to invoke execution of the macro. Macros are very simple. Once a macro is invoked it continues until completion. There are no conditional commands.

Macros can be invoked by third party controllers using the ASCII command, **#MACRO**.

Some system events may have macros attached to them. For example, the integrator may define a macro to be executed when a GPIO input detects a contact closure. Another macro may be defined for when the contact closure ends.

Chapter 4: Overview of Services

The scope of this manual is to provide details on how to integrate a 3rd Party Controllers with a StreamNet IP-Based Media Distribution system.

It is assumed that the user understands **User Datagram Protocol** (**UDP**), TCP and **Unicast** and **Multicast** addressing.

It is also assumed that the user has used the ***StreamNet Dealer Setup*** tool to configure the StreamNet system.

This chapter and the subsequent information in this document explain how to communicate with the StreamNet system and integrate a 3rd Party Controller with a StreamNet system.

COMMUNICATING WITH THE STREAMNET SYSTEM

The StreamNet protocol uses ASCII messages to communicate with services associated with the devices in the StreamNet system. These messages are either control or status messages.

SERVICES

In the StreamNet system, system/device functions are grouped together into logical constructs called Services. A single StreamNet device may host multiple services. All communication with StreamNet is addressed to one or more of these services hosted by the individual devices.

ASCII messages are not addressed to devices but to these virtual services which reside on each device. This scheme alleviates the need to know the IP addresses of every StreamNet device.

Service Names

Every service must have a unique name which can be used to address ASCII messages. Every service may have a room name associated with it. Services may also be assigned to multiple group names. Service names, room names and group names may all be used to address ASCII messages. If a room name is used as an address, all services assigned to that room will react. If a group name is used, all services assigned to that group will react. There is a special group name (All) to which all services will react.

Service Types

The following is a list of service types supported, although not all physical StreamNet devices support all the listed services.

- **Audio Renderer**
- **Audio/Video Renderer**
- **Audio Source**
- **Audio/Video Source**
- **Audio-only Stream Proxy**
- **User Interface (UI)**
- **Intercom**
- **General Purpose IO (GPIO)**
- **Root**

Common Services by Device

The following is not a comprehensive list of services available. To determine the services available for a specific StreamNet application, consult the appropriate device documentation.

Amplifier / Player Device
(ie.: SpeakerLinX SL220, SL250, SL251, SL254, SL9250-CS)

- Audio Renderer
- Audio Source
- Audio-only Stream Proxy
- User Interface (UI)
- Intercom
- Root

Audio / Video Encoder Device
(ie.: MediaLinX MLA100, MLA101, MLAV300, MLA4000, MLA9101-CS, MLAV9300-CS)

- Audio/Video Source
- Audio-only Stream Proxy (MLA100 & MLA101 only)
- Root

Touch Panel
(ie.: TouchLinX TL430, TL700)

- User Interface (UI)
- Intercom
- Root

Legacy Controller Device
(ie.: ControlLinX CL100, CL9100-CS)

- General Purpose IO (GPIO)
- Root

Service Configuration

Each physical StreamNet device may be configured to have some or all of its specific services enabled.

The Root Service is the only service that is always active on all devices.

A Media Server in Room X may have:

- An Audio Renderer Service
- An Audio/Video Renderer Service
- An Intercom Service
- An Audio Source Service
- An Audio/Video Source Service
- Audio-only Local Source Service
- An Audio-only Stream Proxy Service
- A Root Service

An Encoder in Room Y may have:

- An Audio Source Service
- An Audio/Video Source Service
- An Audio-only Local Source Service
- An Audio-only Stream Proxy Service
- A Root Service

An Amplifier in Room Z may have:

- An Audio Renderer Service
- A Root Service

Chapter 5: Services Described

AUDIO/VIDEO RENDERER SERVICE

Overview

A Renderer Service (Audio/Video) is responsible for playing audio and/or video from network sources. It is also responsible for controlling any attached amplifier or display device.

Renderers may support various combinations of the possible media stream types. Some of the media streams currently supported are:

1. Stereo Audio (44.1KHz, 16bit, stereo, linear PCM)
2. S/PDIF Digital Audio (transparent pass-through of S/PDIF bit stream)
3. Uncompressed Video (1 Gbps uncompressed video stream of various resolutions)
4. Compressed Video (less than 100Mbps video stream of various resolutions)

New stream types or variations of these stream types may be added at any time.

A Renderer may create S/PDIF, line level, or amplified audio outputs. Renderers which support video may output HDMI/DVI, component, composite, or S-video.

The Renderer service may provide controls for the output of the media being rendered, such as volume, bass, treble, and Display On/Off.

AV Multi-Zone Operation

Audio and video renderers may be joined together into a “super zone” or “party mode” where they all play the same source. This is called Multi-zone operation. The **#MULTIAUDIO** command is used to control this mode. Renderer services may be forced to “JOIN” or “LEAVE” a multi-zone session. Each multi-zone session has a name specified when it is created. A renderer may only be a member of one multi-zone session.

If one renderer in a multi-zone session changes its source then all renderers in the session will follow. This will continue as long as the renderer service is part of the session. The renderer will drop out of the session if it receives a **#MULTIAUDIO LEAVE** or **#ACTIVE OFF**.

The source selection is the only attribute which will track throughout the multi-zone session. In this way volume and other tone controls may be adjusted for each zone individually.

Commands may be addressed to the SessionName. If volume commands are sent addressed to the session, it is recommended to use **#LEVEL_SET** commands instead of **#LEVEL_UP** and **#LEVEL_DOWN** commands. The current volume levels may be different from zone to zone. The **#LEVEL_SET** will put all of the zones at the same level.

If Do Not Disturb (DND) is enabled then the JOIN command will be ignored.

If Multiaudio is disabled during configuration then the JOIN will be ignored.

After creating the multi-zone session it is recommended to send **#ACTIVE ON**, **#MUTE OFF** and maybe **#LEVEL_SET VOLUME DEFAULT**. The controller may also send a source select command to force the session to have the same audio source as the current room. In this way it doesn’t matter whether the source was selected before or after the session was established. All of these commands would be addressed to the sessionName.

Examples

Audio/Video Renderer Command Details

The following pages detail the functions of many StreamNet ASCII commands.

The format of the text is as follows:

- A brief description of what the command does.
- Command name and format.
- An example of the ASCII message syntax for the command.
- Notes specific to the command and explanation of possible responses generated.

Command: #ACTIVE OFF

Sets the Active Audio Renderer Service to OFF (InActive).

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#ACTIVE OFF**

Command a renderer service to enter the **OFF** state.

The Audio Renderer service Room 1 Player is set to the **OFF** state.

- **NOTE**: In this state, the renderer service will not listen to the audio source. It will also stop all audio output. In effect, if there was an audio source being rendered, the rendering will stop (that is, no music will be heard).

Command: #ACTIVE ON

Sets the Active Audio Renderer Service to ON (Active).

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#ACTIVE ON**

The audio renderer service Room 1 Player is set to the **ON** state.

- **NOTE**: When the renderer service enters the ON state, it will maintain the last settings for the Audio Source, Treble, Bass, Equalizer and Balance. However, Volume will be reset to the default level set during configuration. Mute will be OFF and any multi-zone audio sessions will be dropped.

Command: #AMP OFF

Commands a Renderer Service to turn its Amplifier OFF.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#AMP OFF**

The amplifier associated with the Audio Renderer service Room 1 Player is set to the **OFF** state.

- **NOTE**: Although the amplifier is turned off, if the renderer service is active, it will still be listening to the audio source. Commands can still be issued to the renderer service to change the audio sources and settings.

Command: #AMP ON
Commands a Renderer Service to turn an Amplifier ON.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#AMP ON**

The amplifier associated with the Audio Renderer service Room 1 Player is set to the **ON** state.

- **NOTE**: ***#AMP OFF*** and ***#AMP ON*** have no effect on the renderer settings.

Levels
To control, report, and display analog (i.e., continuous) quantities such as Volume, Balance, Brightness, and so on, the level construct was established. A level is a variable which may have any value between 0% and 100%, inclusive. This percentage is mapped to the actual analog value based on context and this mapping need not be linear.

Example:

A volume of 0% might be an attenuation of infinity, 50% is -25dB, 75% is -15dB, and 100% is 0dB.

For Audio Balance 0% is all left channel, 50% is equal for both channels, and 100% is all right channel.

Command: #EQ_PRESET

Command: #LEVEL_DN BALANCE
Commands a Renderer Service to Shift the Balance Level to the Left Channel.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_DN BALANCE**

Shifts the Balance level of the Audio Renderer service Room 1 Player towards the Left Channel (by Decreasing the Right Channel).

Command: #LEVEL_DN BAND_x
Decreases the Band Level.

- x is the Band number indicator (1 through 5).

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_DN BAND_1**

Commands a Renderer Service to Decrease the specified Band Level. In the above example, Band Level, Indicator 1 is decreased for the Audio Renderer service on the device Room 1 Player.

Command: #LEVEL_DN BASS
Commands a Renderer Service to Decrease the Bass Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_DN BASS**

Decrease the Bass Level of Audio Renderer service Room 1 Player.

Command: #LEVEL_DN TREB
Commands a Renderer Service to Decrease the Treble Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_DN TREB**

Decrease the Trebel Level of Audio Renderer service Room 1 Player.

Command: #LEVEL_DN VOL
Commands a Renderer Service to Decrease the Volume Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_DN VOL**

Decrease the Volume Level of Audio Renderer service Room 1 Player.

Command: #LEVEL_SET BALANCE, x

Commands a Renderer Service to Set the Balance Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_SET BALANCE, x**

Set the Balance Level of Audio Renderer service Room 1 Player based on the value x.

- x is a number between 0 – 100.
- x = 50 will set the balance to equal for both channels.
- x < 50 will shift the balance toward the left channel (by decreasing the right channel).
- x > 50 will shift the balance toward the right channel (by decreasing the left channel).

Command: #LEVEL_SET BAND_x, y

Sets the Band Level to a specific value.

- x is the band indicator (1 through 5).
- y is a number between 0 – 100.
- y = 50 will set the BAND_1 to nominal.
- y < 50 will decrease the BAND_1 level from nominal.
- y > 50 will increase the BAND_1 level from nominal.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_SET BAND_1, y**

Commands a Renderer Service to set the BAND_1 Level.

Set the Band_1 Level of Audio Renderer service Room 1 Player based on the value of y.

Command: #LEVEL_SET BASS, x

Commands a Renderer Service to Set the Bass Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_SET BASS, x**

Set the Bass Level of the Audio Renderer service Room 1 Player based on the value x.

- x is a number between 0 – 100.
- x = 50 will set the bass to nominal.
- x < 50 will decrease the bass from nominal.
- x > 50 will increase the bass from nominal.

Command: #LEVEL_SET TREB, x

Commands a Renderer Service to Set the Treble Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_SET TREB, x**

Set the Trebel Level of the Audio Renderer service Room 1 Player based on the value x.

- x is a number between 0 – 100.
- x = 50 will set the treble to nominal.
- x < 50 will decrease the treble level from nominal.
- x > 50 will increase the treble level from nominal.

Command: #LEVEL_SET VOL, x

Commands a Renderer Service to Set the Volume Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_SET VOL, x**

Set the Volume Level of the Audio Renderer service Room 1 Player based on the value x.

- x is a number between 0 – 100.
- x = 0 will set the volume to the minimum level.
- x =100 will set the volume to the maximum level.

Command: #LEVEL_UP BALANCE

Commands a Renderer Service to Shift the Balance Level to the Right Channel.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_UP BALANCE**

Shifts the Balance Level of the Audio Renderer service Room 1 Player towards the Right Channel (by decreasing the Left Channel.)

Command: #LEVEL_UP BAND_x

Increases the device Equalizer Band Level.

- x is the band indicator (1 through 5).

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_UP BAND_1**

Commands a Renderer Service to Increase the BAND_1 Level.

Increases the Band Level, Indicator 1 for the Audio Renderer service Room 1 Player.

Command: #LEVEL_UP BASS

Commands a Renderer Service to Increase the Bass Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_UP BASS**

Increases the Bass Level of the Audio Renderer service Room 1 Player.

Command: #LEVEL_UP TREB

Commands a Renderer Service to Increase the Treble Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_UP TREB**

Increases the Trebel Level of the Audio Renderer service Room 1 Player.

Command: #LEVEL_UP VOL

Commands a Renderer Service to Increase the Volume Level.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#LEVEL_UP VOL**

Increases the Volume level of the Audio Renderer service Room 1 Player.

Command: #MENU_LIST m, n, sources

Gets a List of Audio Sources available to the specified Renderer Service.

Example: **#@Room 2 Player:~UDP10.15.0.11_5000#MENU_LIST 1,6,SOURCES**

- **NOTE**: m,n specifies the range of items to be listed from the list of sources. m = item number of the first item on the menu list to be listed, through n = the last item in the range.

Example: m=1 and n=6, will show the first through the sixth item, or m=3 and n=5 will show the third through the fifth items in the menu list.

- **NOTE**: The command is requesting a list of six audio sources associated with this Service. The response can come in a series of messages depending on the numbers of audio sources associated with the Room 2 Player.

Responses Received

First Response

#@TL3800538001161007532_1~UDP10.15.0.11_2141:Room 2 Player#MENU_RESP
{{<source id="myXM" children="0" itemnum="1" idpath="sources" disppath="sources" display="myXM" type="audio/source" />}}

- **id** = "myXM"
- "**myXM**": Service name of the audio sources. Any commands issued to this audio source must be addressed to its Service name "myXM".
- **children** = "0": indicates a terminal node (no more lists under this audio source).
- **itemnum** = "1": indicates the item number on this menu. 1 indicates this is the first item on the menu. If this is a positive number, then there are more audio sources in the list. A negative number indicates the end of the list.
- **idpath** = "sources": If there are further list(s) under this MENU, then the IDPath concatenated with the Source ID defines the path for the next item on the menu.
- **dispath** = "sources": This string can be used to display the current menu path, if desired.
- **display** = "myXM": This string can be used to display this menu item. (based on itemnum).
- **type** = "audio/source": only audio/source serviceType commands may be issued to this service.

The remaining responses show four more audio sources and can be parsed similarly:

Second Response

#@TL3800538001161007532_1~UDP10.15.0.11_2141:Room 2 Player#MENU_RESP
{{<source id="Stream 3" children="0" itemnum="2" idpath="sources" disppath="sources" display="Stream 3" type="audio/source" />}}

Third Response

#@TL3800538001161007532_1~UDP10.15.0.11_2141:Room 2 Player#MENU_RESP
{{<source id="Local Source 2" children="0" itemnum="3" idpath="sources" disppath="sources" display="Local Source 2" type="audio/localsource" />}}

Fourth Response

#@TL3800538001161007532_1~UDP10.15.0.11_2141:Room 2 Player#MENU_RESP
{{<source id="Stream 1" children="0" itemnum="4" idpath="sources" disppath="sources" display="Stream 1" type="audio/source" />}}

Fifth Respone

#@TL3800538001161007532_1~UDP10.15.0.11_2141:Room 2 Player#MENU_RESP
{{<source id="Stream 2" children="0" itemnum="5" idpath="sources" disppath="sources" display="Stream 2" type="audio/source" />}}

Sixth Response

#@TL3800538001161007532_1~UDP10.15.0.11_2141:Room 2 Player#MENU_RESP
{{<sources idpath="sources" itemnum="-1" />}}

- **NOTE**: **itemnum**="-1". The "**-**" sign indicates that there are no more audio sources.

  If **itemnum** = a positive number, then there are more audio sources assigned to this Service and the command needs to be re-issued until **itemnum**="-x" is received.

Example: **#@Room 2 Player:~UDP10.15.0.11_5000#MENU_LIST 7,9,SOURCES** will get a response detailing the seventh through ninth items from the MENU_LIST.

- **NOTE**: If there are only five items in the menu (as in this case), then a command requesting a list of ten sources, like:

  #@Room 2 Player:~UDP10.15.0.11_5000#MENU_LIST 1,10,SOURCES

  will still generate the exact same response as above, with the sixth response showing "**itemnum**=-1".

Command: #MENU_SEL {{path}}

Selects the Audio Sources from a list available to the Renderer Service.

Example: **#@Room 2 Player:~UDP10.15.0.11_5000#MENU_SEL {{sources>Stream 2}}**

This command selects Stream 2 as the source to be rendered.

- **NOTE**: The menu path for this list is established by concatenating the item's idpath and source id separated by the "**>**" symbol.

Example: {{sources>Stream 2}} becomes the menu path to select the fifth (itemnum=5) audio source based on the responses received from the **MENU_LIST** example above.

- **NOTE**: Alternatively, in this specific case, an **#SRC_SEL** command (see below) may be used.

Command: #MULTIAUDIO

This command causes the renderer service to join an AV super-zone referred to as a "multi-zone session". (see section: AV Multi-Zone Operation)

Command: #MULTIAUDIO JOIN {{sessionName}}

sessionName is the name of the multi-zone session. This name may be any string which would be a valid StreamNet group address name; but it should not be an existing group name because a new group will be formed using this name. Using an existing group name would have the effect of making the existing group members part of the multi-zone session and vice versa.

When the addressed renderers receive a JOIN command they will automatically leave any currently active session to join the new session. Also if the renderer is in the active="0" state then it will change to the active="1" state before joining the session.

Command: #MULTIAUDIO LEAVE

When the addressed renderers receive a LEAVE command they will immediately leave any currently active session.

Command: #MUTE OFF
Commands a Renderer Service to set Mute to the OFF state.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#MUTE OFF**

Command: #MUTE ON
Commands a Renderer Service to set Mute to the ON state.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#MUTE ON**

Command: #MUTE TOGGLE
Commands a Renderer Service to Toggle the state of Mute.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#MUTE TOGGLE**

Command: #QUERY renderer
Query the settings for current Audio Source being Rendered.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#QUERY renderer**

- **NOTE**: A **#QUERY** command is interrogative and gets a declarative response **#REPORT**.

  The **#QUERY RENDERER** gets a report indicating the current settings for the Audio Source being rendered.

Sample response:

#@SL22004280001161000057_4~UDP10.15.0.11_4881:Room 1 Player#REPORT {{<report type="state" vol="29" balance="50" bass="50" treb="50" loud="0" mute="0" audioSession="All Rooms Multiroom" audioSessionActive="1" ampOn="1" />}}

Parsing Reports

- **NOTE**: Even though the format looks like XML, it isn't. The report will always be flat (i.e., no nested <> brackets).

The order of the attributes may change. Some may be added or omitted in any given **#REPORT** message. Parsing of the message should key off the attribute name. For example, for "balance", a suggested parsing method would be to scan for "=". The attribute name is to the left of "=" and the value is to the right. The fromAddress (i.e., Room 1 Player) should be used to qualify the report before accepting the other data attributes (i.e., Room 1 Player). Validate the origin of the message before storing or using the reported status.

- **NOTE**: The data may be distributed among multiple **#REPORT** messages even if the report(s) is the result of a single #QUERY message.

Command: #QUERY CURRENT_SOURCE
Queries the current Audio Sources being Rendered.

Example: **#@Room 2 Player:~UDP10.15.0.11_5000#QUERY CURRENT_SOURCE**

Response Received:

#@TL3800538001161007532_1~UDP10.15.0.11_2230:Room 2 Player#REPORT {{<report type="state" currentSource="myXM" currentSourceIP="10.15.96.149" permId="MLA10105151001161008019_1" />}}

- **NOTE**: Room 2 Player is currently rendering an Audio Source called "myXM".

Command: #SET

Command: #SLEEP
Commands a Renderer Service to set a specific device to the Sleep state.

Example: **#@Room 1 Player:~UDP10.15.0.11_5000#SLEEP**

Sets the Room 1 Player device to Sleep.

Command: #SRC_SEL
De-selects all Audio Sources from Renderer.

Example: **#@Room 2 Player:~UDP10.15.0.11_5000#SRC_SEL " "**

- **NOTE**: If no Audio Source is specified between the " ", the current Audio Source will be deselected resulting in no Audio Source being rendered by the service: Room 2 Player.

Command: #SRC_SEL {{source ID}}
Selects a specific Audio Source to be played.

Example: **#@Room 2 Player:~UDP10.15.0.11_5000#SRC_SEL {{Stream 3}}**

Selects Stream 3 to be rendered by service Room 2 Player.

- **NOTE**: The source name is case sensitive and must be bound by the double curly brackets, as in **{{Stream 3}}**. The double curly brackets are required since the Service name has a space (non alpha-numeric character)

Command: #TEMPSRC {{sourceName}} [,timeOut value]
Forces the renderer service to temporarily select the specified source to be played. The time out value (in seconds) is optional. When the **#TEMPSRC** command is canceled or times out the renderer returns to the original source. If a new **#TEMPSRC** command comes in while another is still active the new command takes effect. Nested **#TEMPSRC** commands are not supported; the first **#TEMPSRC** is canceled by the second. And when the second **#TEMPSRC** is canceled, the renderer will return to the original source.

Example: **#TEMPSRC {{camera1}}, 60**

Will switch renderer to "camera1" for 60 seconds and then switch back to whatever source it was rendering before.

Command: #TEMPSRC {{sourceName}}, OFF
If there is an active **#TEMPSRC** command and the temporary source has the specified name then the **#TEMPSRC** command will be canceled and the renderer service will return to the original source. If the source name does not match the command will have no effect. In this way a **#TEMPSRC** command may be cancelled without fear of interfering with a different **#TEMPSRC** command which may have subsequently overridden it.

Command: #TEMPSRC OFF
Will unconditionally cancel any active **#TEMPSRC** command.

AUDIO/VIDEO SOURCE AND SOURCE PROXY SERVICE

Overview

An Audio/Video Source service is responsible for taking in audio and video from some source and encoding them into IP-based data packets and transmitting these packets over the network. The Source service may also provide some control over the source device.

All current StreamNet AV Source services are controlled in more less the the same way. Typically all control functionality is defined by the StreamNet Soft Adapter (SSA) which gets installed when the device is configured. Some source services may provide IR control of the attached AV device. Some source services may provide serial control of the attached AV device with the appropriate SSA. IP control of the AV device resides entirely with the SSA. Or control of the attached AV device may be provided directly by a third party controller.

The primary difference between an AV Proxy service and other AV Source services is that the Proxy service gets the audio and/or video over IP (possibly from the internet); and the other source services capture the audio/video on physical connections such as S/PDIF or HDMI.

The Audio/Video Source Proxy Service represents (or proxies) network media sources which do not conform to the StreamNet protocol (non-StreamNet devices). For example, an Audio/Video Proxy Service will convert the content and control commands of a 3rd Party media server to be compatible with the menus and other commands of the StreamNet ASCII protocol.

An Audio/Video Proxy Service must have a StreamNet Source Adapter installed to browse the source content and to initiate the media stream. The Source Adapter translates various media servers' protocol or internet radio services' protocol into standard StreamNet protocol.

The Audio/Video Source service may also provide controls (such as IR or RS-232) for management of the legacy AV source device.

Examples

Command: #MENU_LIST m, n, {{path}}

Requests a List of Media associated with a Media Server.

Example: **#@Stream 1:~UDP10.15.0.11_5000#menu_list 1,10,media**

- **NOTE**: This message is addressed to Stream 1. The command is requesting menu items 1 to 10 of the media associated with this service. Media is the default path name for the first menu list.

Responses Received

First Response

#@SL22004285001161000172_4~TCP10.15.0.11_4750:Stream 1 #MENU_RESP {{<item idpath="media" disppath="media" itemnum="1" id="All Songs" display="All Songs" children="1078" />}}

Example: **{{media>All Songs}}** becomes the next menu path.

- **itemnum = "1"** A non-zero number indicates this is the first item on this menu list.
- **display = "All Songs"** is the display name for this list.
- **children = "1078"** indicates that this is not a terminal node (i.e., there is another menu list below this level).

Second Response

#@SL22004285001161000172_4~TCP10.15.0.11_4750:Stream 1#MENU_RESP {{<item idpath="media" disppath="media" itemnum="2" id="Artists" display="Artists" children="63" />}}

The next level menu path for this list is established by concatenating the item's idpath and ID separated by the ">" symbol.

Example: **{{media>Artists}}** becomes the next menu path.

- **itemnum = "2"** indicates this is the second item on this menu list.
- **display = "Artists**" is the display name for this list.
- **children = "63"** indicates that this is not a terminal node. ie: there is another menu list below this level.

Responses 3 through 5

These responses can be similarly parsed.

- #@SL22004285001161000172_4~TCP10.15.0.11_4750:Stream 1 #MENU_RESP {{<item idpath="media" disppath="media" itemnum="3" id="Albums" display="Albums" children="116" />}}
- #@SL22004285001161000172_4~TCP10.15.0.11_4750:Stream 1 #MENU_RESP {{<item idpath="media" disppath="media" itemnum="4" id="Genres" display="Genres" children="12" />}}
- #@SL22004285001161000172_4~TCP10.15.0.11_4750:Stream 1 #MENU_RESP {{<item idpath="media" disppath="media" itemnum="5" id="Playlists" display="Playlists" children="3" />}}

Sixth Response

#@SL22004285001161000172_4~TCP10.15.0.11_4750:Stream 1 #MENU_RESP {{<item idpath="media" disppath="media" itemnum="-1" />}}

itemnum = "-1". The negative value indicates there are no more items on this list.

Summary

The service Stream 1 menu has five items which are displayed as:

- **All Songs**
- **Artists**
- **Albums**
- **Genre**
- **Playlists**

Each of these branches has children (another menu list) under them.

Command: #MENU_LIST m, n, {{path}}

This command allows the user to traverse down a Menu List.

As explained above, the next level menu path is established by concatenating the item's idpath and its id separated by a "**>**" character.

Example: **{{media>All Songs}}** becomes the next menu path:

#@Stream 1:~UDP10.15.0.11_5000#menu_list 1,5,{{media>All Songs}}

The command is requesting menu items 1 to 5 of the menu list below the list **All Songs**, which was the first item on the previous menu list.

Responses Received

First Response

#@SL22004285001161000172_4~TCP10.15.0.11_4781:Stream 1 #MENU_RESP {{<song idpath="media>All Songs" disppath="media>All Songs" itemnum="1" id="1481" display="Brown, Crane, Ker, Finesilver" children="0" />}}

The next level menu path for this list is established by concatenating the item's idpath and id separated by a "**>**". Example: {{media>All Songs>1481}} becomes the next menu path for Borwn, Crane, Ker, Finesilver

- **itemnum="1"** indicates this is the first item on this menu list.
- **display="Brown, Crane, Ker, Finesilver"** for this item.
- **children = "0"**, indicates that this may be a terminal node (i.e., there are no items below this level). Therefore, a further #MENU_LIST command would neither get a response nor a reaction. Instead, as explained below, a #MENU_SEL command must be issued.

Responses 2 through 5

These responses can be similarly parsed.

- #@SL22004285001161000172_4~TCP10.15.0.11_4781:Stream 1 #MENU_RESP {{<song idpath="media>All Songs" disppath="media>All Songs" itemnum="2" id="1648" display="(Da Le) Yaleo" children="0" />}}
- #@SL22004285001161000172_4~TCP10.15.0.11_4781:Stream 1 #MENU_RESP {{<song idpath="media>All Songs" disppath="media>All Songs" itemnum="3" id="1606" display="(Hidden Track)" children="0" />}}
- #@SL22004285001161000172_4~TCP10.15.0.11_4781:Stream 1 #MENU_RESP {{<song idpath="media>All Songs" disppath="media>All Songs" itemnum="4" id="1974" display="11 O'Clock Tick Tock" children="0" />}}
- #@SL22004285001161000172_4~TCP10.15.0.11_4781:Stream 1 #MENU_RESP {{<song idpath="media>All Songs" disppath="media>All Songs" itemnum="5" id="1605" display="2000 Miles" children="0" />}}
- **itemnum** = "5". The positive value indicates there are more items on this list.
- **children** = "0", in each of these items indicates that there are no more items under these items.

Summary

"All Songs" (the first item on the menu) has a list of items 1 to 5 which are displayed as:

1. Brown, Crane, Ker, Finesilver
2. (Da Le) Yaleo
3. (Hidden Track)
4. 11 OClock Tick Tock
5. 2000 Miles

The “All Songs” menu had shown it had 1078 children of which the command **#menu_list 1,5,{{media>All Songs}}** requested a list of items 1-5.

Similarly, if the integrator wanted to see the list under the second item, “Artists”, the command syntax would be:

#@Stream 1:~UDP10.15.0.11_5000#menu_list 1,5,{{media>Artists}}

Command: #QUERY SOURCE

Gets details on the current Media being played.

Example: **#@Stream 1:~UDP10.15.0.11_5000#QUERY source**

Response Received

#@SL2200428500116100017 2_4~UDP 10.15.0.11_4028:Stream 1#REPORT {{<report type=”source” artwork=”http://10.15.100.116/7.0.5/Database/Music/Covers/s4/s3/Nails-Corpus Christi” display=”song/artist/album/genre” song=”14 Dreams” album=”Corpus Christi” artist=”Nails” genre=”Rock” time=”376” percent=”59” sngPlIndex=”79” sngPlTotal=”11169” source=”Stream 1” elapsed=”223744”
next=”14 Years” active=”2” controlState=”PAUSE” shuffle=”0” />}}

- This command gives a report of the current song in “Stream 1”
- The display attribute gives a suggestion of which items to display and with what priority.
- In this example, the audio source “Stream 1” is suggesting that the song title is the most important item to display, followed by the artist’s name, the album name and finally the genre.
- The UI device is not bound by the suggestion.

▶ **NOTE**: **#REPORT** responses shall in no case exceed 1000 characters, but data may be distributed among multiple **#REPORT** messages.

USER INTERFACE (UI) SERVICE

Overview

A UI service handles all of the configuration and control for a particular user interface. Some common functionality provided by UI services are:

1. Converting key press commands to other ASCII commands.
2. Converting IR codes to key press commands or other ASCII commands.
3. Forwarding commands to a subordinate UI component such as a keypad or Flash GUI.

Unlike most other service types, UI services tend to have differing functionality from device to device.

INTERCOM SERVICE

Overview

Behind the scenes, the operation of the intercom service is very complex; but for basic intercom operation only a few commands are required.

An Intercom session is like a conference call between two or more intercom zones. More than one simultaneous session may exist for any zone. For simplicity, the StreamNet Intercom service completely manages multiple Intercom sessions; the controller need not worry about the possibility of multiple sessions. It is possible for the controller to take complete control of the Intercom sessions but that will not be covered here.

Currently there are three kinds of Intercom Sessions:

A. Station to Station(s) Session

This session type allows two-way communication between two or more intercom stations. When a user presses the Push-to-Talk button their voice is transmitted to the other stations in the session. Communication is half-duplex (Only one station may Transmit at a time.) This session will time out after a configurable time if there is no activity.

B. Room Monitoring Session

This session type allows stations in one or more rooms to listen to (monitor) another room. This not very different from a Station to Station session except that in the room being monitored the microphone is on by default. Stations can talk to the room being monitored by pressing their Push-to-Talk buttons. This session will stay active until terminated.

C. Door Answering Session

This type of session is initiated when the DoorBell button of a DoorLinX is pressed. This session is very similar to the Room Monitoring session. The Door station is room being monitored. The DoorLinX will play an announcement MP3 file at the beginning of the session. This session will time out after after a configurable time if there is no activity.

Station to Station

To start a Station to Station Session send the following command to the zone which is initiating the call:

#INTERCOM CREATE, “ID”, “audience”

This message is sent by a user interface or third party controller to an intercom service causing the intercom service to create an intercom session between itself and the specified audience.

“ID” is the name of the session. It may be any string but should usually be the same as the “audience”. This is the name of the session that should be displayed on the User Interface.

“audience” is the service, room or group whose participation is desired. The “audience” is an ASCII protocol “ToAddress” which is used internally by the IC service to send out commands to start the session.

To turn on the microphone for a particular zone send the following command addressed to the zone:

#INTERCOM PTT

This command will turn on the microphone and keep it on for 300ms. This command must be repeated at 100ms intervals to keep the microphone on.

When the user is done speaking the following command should be sent:

#INTERCOM PTT OFF

This will immediately turn off the microphone. This is better than waiting for the time out. It is possible to direct a PTT command to any specific CURRENT session by adding the sessionTag.

A PTT command without any qualifier always transmits to the RECENT session. If there are no CURRENT sessions then an unqualified PTT will create a new default session.

Normally, it is best to let the Station to Station session time out on its own; however, if desired, the session may be terminated by sending the command:

#INTERCOM LEAVE [, tagNumber]

Where tagNumber is a unique identification number for an Intercom session. The tag number is assigned automatically by the system when the session is created. The tag numbers of all sessions are available in the status report of the intercom service.

If the tag number is omitted all of the sessions will be dropped by the service(s) addressed, station-to-station, monitor and door sessions. Leave commands will almost always be addressed to a room name.

- **NOTE:** It is not required to send a leave command for station-to-station or door sessions. The session will quickly time out if there is no activity. The Leave command is usually sent in response to a user action to cancel current sessions.

The command will terminate all Intercom sessions in that zone.

Intercom Monitor Session

A room monitoring session allows a user to listen to a remote room from one or more other rooms. To start a Room Monitoring Session send the following command addressed to the zone to be monitored:

#INTERCOM MONITOR, “roomName”, “audience”

“roomName” is the name of the room which is to be monitored.

“audience” is a service, room, or group which will monitor the specified room.

Note that if there is an existing session monitoring the room then this command ADDS the new audience to the existing session.

To terminate a room monitoring session in any zone(s) send the following command addressed to the zone(s):

#INTERCOM LEAVE [,MONITOR]

This message can be sent addressed to a single room to stop that one zone from monitoring or addressed to the room being monitored to terminate the entire session.

Intercom Entry Sessions

Normally a Entry [answering] Session is initiated automatically by a DoorLinX device when a guest presses the doorbell button. The “audience” of this Entry session is configurable using the system set-up tool.

The DoorLinX initiates an Entry session with all of the rooms [services] in its “audience”. It begins by playing a stored MP3 file which typically is a chime but could be any MP3 file. After the first room speaks to the door station all of the other rooms in the audience are dropped from the session. The microphone at the door is left open so that the guest may answer.

An intercom room may be forced out of a Door Session by sending a Leave command:

#INTERCOM LEAVE [, ENTRY]

#INTERCOM LEAVE [, tag]

If “ENTRY” or “tag” is left off the LEAVE command, all intercom sessions in the addressed room will be terminated.

Status Report with a Current Intercom Session

The following is a typical status report for a station-to-station intercom session:

#@~STATUS:Room1 Intercom SL#REPORT {{<report type=”intercom” myServiceName=”Ryan’s Room Intercom SL” serviceType=”intercom” myRoomName=”Ryan’s Room” sessions=”1” active=”1” recentSessionID=”INTERCOM” recentSessionTag=”00000088” icVol=”84” privacy=”0” DND=”0” kind=”conversation” monitor=”0” entry=”1” initiator=”Connor’s Room” />}}

Status Report with No Current Intercom Sessions

#@~STATUS:Ryan’s Room Intercom SL#REPORT {{<report type=”intercom” myServiceName=”Ryan’s Room Intercom SL” serviceType=”intercom” myRoomName=”Ryan’s Room” sessions=”0” active=”0” recentSessionID=”None” recentSessionTag=”FFFFFFFF” icVol=”84” privacy=”0” DND=”0” kind=”none” monitor=”0” entry=”1” initiator=”” />}}

GENERAL PURPOSE IO (GPIO) SERVICE

Overview

The General Purpose IO (GPIO) service can be thought of as a “Miscellaneous Function” or a catch-all type Service. Services that don’t neatly fit into any of the other services described may often be found under this umbrella service type. Some of the common I/O types associated with GPIO services are IR, RS232 serial, and relay contact closures.

Many GPIO services have LUA interpreters associated with them. This allows custom drivers to be written to perform any functionality desired. See Appendix for commands to control specific devices.

ROOT SERVICE

Overview

Every device has exactly one root service. The root services manage common and miscellaneous housekeeping functions for each device. The root service is always active, even when all of the other services on the device have been disabled.

Appendix: SpeakerLinX

The following SpeakerLinX devices will respond only to the specified commands as outlined below.

Commands must be issued to the appropriate serviceType specifying the serviceName in the **:toAddress** field.

SPEAKERLINX AUDIO RENDERER COMMANDS

SpeakerLinX Audio Renderer Service Commands								
ASCII STRING	COMMENTS	Device						
		SL220	SL250	SL251	SL254	SL9250-CS	SN1001
#ACTIVE OFF	Enter the INACTIVE state	✓	✓	✓	✓	✓	✓
#ACTIVE ON	Enter the ACTIVE state	✓	✓	✓	✓	✓	✓
#AMP OFF	Turn OFF the amplifier	✓	✓	✓	✓	✓	✓
#AMP ON	Turn ON the amplifier	✓	✓	✓	✓	✓	✓
#LEVEL_DN BALANCE	Shift balance to the Left	✓	✓	✓	✓	✓	✓
#LEVEL_DN BAND_x	Decrease band level to a specific value	✓	✓	✓	✓	✓	✓
#LEVEL_DN BASS	Turn down bass level	✓	✓	✓	✓	✓	✓
#LEVEL_DN TREB	Turn down treble level	✓	✓	✓	✓	✓	✓
#LEVEL_DN VOL	Turn down volume level	✓	✓	✓	✓	✓	✓
#LEVEL_SET BALANCE, x	Set the balance level, 0 = Full Left / 100 = Full Right	✓	✓	✓	✓	✓	✓
#LEVEL_SET BAND_x, y	Set the Equalizer Band to a specific level (0-100%)	✓	✓	✓	✓	✓	✓
#LEVEL_SET BASS, x	Set the bass to specific level, (0-100%)	✓	✓	✓	✓	✓	✓
#LEVEL_SET TREB, x	Set the treble to specific level, (0-100%)	✓	✓	✓	✓	✓	✓
#LEVEL_SET VOL, x	Set the volume to specific level, (0-100%)	✓	✓	✓	✓	✓	✓
#LEVEL_UP BALANCE	Bump balance to the Right	✓	✓	✓	✓	✓	✓
#LEVEL_UP BAND_x	Bump the equalizer band setting up	✓	✓	✓	✓	✓	✓
#LEVEL_UP BASS	Bump up bass level	✓	✓	✓	✓	✓	✓
#LEVEL_UP TREB	Bump up treble level	✓	✓	✓	✓	✓	✓
#LEVEL_UP VOL	Bump up volume level	✓	✓	✓	✓	✓	✓
#MENU_LIST m, n, sources	List of sources available to the renderer service	✓	✓	✓	✓	✓	✓
#MENU_SEL {{path}}	Make a Menu Selection	✓	✓	✓	✓	✓	✓
#MULTIAUDIO	Force the Renderer into a Multizone audio session	✓	✓	✓	✓	✓	✓
#MUTE OFF	Set MUTE=OFF	✓	✓	✓	✓	✓	✓
#MUTE ON	Set MUTE=ON	✓	✓	✓	✓	✓	✓
#MUTE TOGGLE	Toggle the Mute setting	✓	✓	✓	✓	✓	✓
#QUERY CURRENT_SOURCE	Query which source is being rendered	✓	✓	✓	✓	✓	✓
#QUERY RENDERER	Query for the current renderer settings	✓	✓	✓	✓	✓	✓
#SET		✓	✓	✓	✓	✓	✓
#SLEEP		✓	✓	✓	✓	✓	✓
#SRC_SEL	Command renderer service to stop listening to any source	✓	✓	✓	✓	✓	✓

SpeakerLinX Audio Renderer Service Commands							
ASCII STRING	COMMENTS	Device					
		SL220	SL250	SL251	SL254	SL9250-CS	SN1001
#SRC_SEL {{source id}}	Select a specific source to be rendered	✓	✓	✓	✓	✓	✓
#SRC_SEL NEXT	Choose next available source to be rendered	✓	✓	✓	✓	✓	✓
#TEMPSRC		✓	✓	✓	✓	✓	✓

INDEX

Symbols

A

B

D

F

G

I

K

L

M

Q

R

S

T

U

V

GLOSSARY OF TERMS

A

Argument - Most often an optional fields that proceed keywords. When used, there are one or more spaces between the keyword and the first argument and a comma is used to separate each successive argument. Arguments can be in the format of Meaningful Strings, Character Literals, Numeric Scalars or XML or other Special Formats.

ASCII Protocol - The basic control and status reporting mechanism of the services in StreamNet. It consists of various audio and video messages and the respective parameters represented in strings of ASCII characters. These messages also contain the associated routing and administrative information used to inform recipients of the destination and origin of the messages or provides various supplemental information.

Audio/Video Renderer Service - One of several Services associated with devices in the StreamNet system. The Audio/Video Renderer service

Audio/Video Source Service - One of several Services associated with devices in the StreamNet system. The Audio/Video Source service is responsible for taking an analog or digital audio and/or the corresponding video signal from a legacy device (CD or DVD Changer), encoding them into IP-based data packets and transmitting them over the network.

B

C

Case Sensitivity - Case Sensitivity is a term used to describe the actions of a parsing scheme that is sensitive to the use of uppercase and lowercase characters and interprets a string of characters differently based on the case of the characters in the string.

Case Sensitivity comes into play in a StreamNet system when, for instance, the Source or Renderer service name is used as an argument in a command line: Service name: Room 2 Player

Correct Command Line - the Service name is used as an argument and is correctly entered:

#@Room 2:~UDP10.15.0.11_5000#query service {{Room 2 Player}}

Incorrect Command Line - the Service name is used as an argument but is incorrectly entered:

#@Room 2:~UDP10.15.0.11_5000#query service {{room 2 player}}

When the source or renderer service name is not used as the toAddress, it is not case sensitive:

#@room 2 player:~UDP10.15.0.11_5000#query renderer

Character Literals - Argument labels that are comprised of character strings which do not need any special delimiters and contain no blanks, spaces, commas or other special, non-alphanumeric characters. Arguments that do contain special characters must be enclosed within double brackets "**{{...}}**".

Command - A string of ASCII characters that form a valid StreamNet statement which is sent to a device or service to perform a requested action or generate an appropriate response.

D

Declarative - A StreamNet ASCII message command used to report information requested from an unsolicited status report (periodic or caused by a change in a device's state) or via a response to an interrogative message. The most common StreamNet declarative is the #REPORT message.

E

F

Flash - A storage method/device for digital media.

fromAddress - Indicates which entity sent the message and is used to address any response that may be generated by the message. If a response is expected, then the fromAddress should specify the address the response should be sent to. The colon symbol "**:**" must precede all ***fromAddress*** fields.

G

General Purpose IO (GPIO) Service - The General Purpose IO service is responsible for sending IR or RS-232 commands to devices connected to a ControLinX. The GPIO service can be thought of as a "Miscellaneous Function" or catch-all type service category for services that don't fit neatly into any of the other services described.

H

I

IP Address - A network IP Address is the network address assigned to each device in a StreamNet system and is required for sending or receiving StreamNet ASCII commands for device operation as well as device status inquiry.

Imperative - A type of ASCII message command that initiates an action or directly causes a change in state of a specific device.

INTERCOM Service - An Intercom Service provides one-way (Unicast) or two way (Multicast) voice communication with other Intercom Services.

Interrogative - An ASCII message command used to request information from a particular device.

IR - A method of remotely controlling a device without having to be physically connected to it by sending specific commands using wireless Infrared (IR) technology.

J

K

KEY - A UI/Keypad/IR service command used to identify the specific function key that was pressed in order to initiate an action based on the depressed key.

Keyword - The keyword is a required field in all ASCII command statements and specifies the action to be executed. The keyword field must be preceded by the "**#**" symbol with no intervening spaces. The keyword is one word and contains no spaces, commas or other special characters except the underline "**_**" character.

L

M

MENU - A set of StreamNet ASCII messages which makes use of all three ASCII message types: Interrogative, Declarative and Imperative. The MENU ASCII messages are #MENU_LIST, #MENU_RESP and #MENU_SEL.

Multicast - A One-to-Many communication technique where all parties can communicate with each other over an IP network infrastructure, often in real-time.

Multiplexer - A device used to combine and send a stream of data over a communication line. The multiplexer will also separate a received data stream into component packets.

N

Numeric Scalars - An argument format style in which the argument must be in the format of a decimal number.

O

P

Point-To-Point - A method of communication between one sender and one receiver over a network - similar to Unicast.

Push To Talk (PTT) - An Intercom Service parameter for the INTERCOM command used to create and start a PTT session, stop a PTT session, monitor a PTT session, enter a PTT session or simply create without starting a PTT session. A PTT session is an Intercom session where the microphone is off unless the user manually pushes the microphone button to talk.

Q

QUERY - Multi-service ASCII command that requests information from a device or service and generates a report based on the feedback received.

R

Root Service - One of several services associated with devices in the StreamNet system. The Root service is the only service that is always active on all devices.

S

serviceName - A unique name assigned to each service in a StreamNet system. The serviceName is then used to address ASCII messages. Each physical StreamNet device may be configured to have some or all of its specific services enabled.

StreamNet - A technology that provides elegant solutions for streaming media & control applications such as digital signage, distribution of HD video and audio, LAN Cloud Matrix Switching™, and audio paging over data networks.

SubNode - A Subnode address is a part of the ASCII command address and can be a part of the TO or FROM address. A subnode address is a special address modifier which instructs the node where to forward the command once it is received. Typically, the subnode address indicates that the service should forward the command to a non-StreamNet device.

T

tag - A 32-bit Intercom service session number that will be used to identify the session in the header of the Intercom audio data and to reference it in other commands.

talkIpAdr - The IP Address to which a device should send replies to - may be Multicast or Unicast.

talkPort - The port to which the device should send replies to.

toAddress - Specifies the name of the service, group or room to which the message is sent. The toAddress must be immediately preceded by the "**@**" symbol. The Address field consists of two parts: the **node** address and the **subnode** address.

U

UniqueID (UID) - The UID of a service is the unchangeable identifier for the service. The UID is based on the combination of the device's serial number and an integer that identifies the specific service on the device. In general, there is no reason to use the UID as all services should be assigned names and all of the user's commands should be addressed to those service names (or room/group names).

Unicast - A method of communication between one sender and one receiver over a network - see Point-To-Point.

User Data Protocol (UDP) - The methods and standards used to send StreamNet ASCII messages over the Internet to any StreamNet and StreamNet-compatible device. If addressed correctly, the message will be routed to the desired service(s).

User Interface (UI) Service - A UI Service provides services for User Interfaces. There are currently two types of UI services: UI/GUI and UI/Keypad. A TouchLinX hosts a UI/GUI service and a SpeakerLinX hosts a UI/Keypad service.

V

W

X

Y

Z