Icon API

System Software Build 1.1.0 (16)

Serial/SSH Overview

To use the rbshell interface Commands take the form:

actor command parameters

You can get a list of actors by hitting the tab key at the command prompt. You can also complete any command by hitting the tab key. If multiple commands are possible then a list of the possible commands will be displayed.

To get help on any command put a question mark after it like this:

SysAdmin getIPv4Config ?

You can get the list of commands for any actor by putting a question mark after the actor name like:

SysAdmin ?

All commands are case insensitive. Commands will show the official case when a tab completes them, but uncompleted commands will work as well. The parameters are evaluated as a series of comma separated JSON elements, so structures and arrays can be entered as JSON.

LifeSize®
A division of Logitech

REST Overview

The Switchboard Rest API provides a simple way to make calls into a switchboard environment. Using HTTP GET and POST requests accesses the API. Several URLs are provided for different types of requests. Either HTTP or HTTPS transport can be used.

Input and output to the API is in the form of JSON data. This data makes up the output body of GET requests, and both the input and output bodies of POST requests. The ordering of fields within the output JSON data is not guaranteed and is unimportant in the input data.

The URL paths and their functions are summarized in the following table.

path	function
/rest/new	Create a new session
/rest/poll/<session-id>	Do a fast poll for events on a session
/rest/request/<session-id>	Call a function within a session
/longpoll.rb?session=<session-id>&timeout=<seconds>	Perform a long poll on a session

Creating a session

A GET request to the path /rest/new will return a JSON structure with a new session ID. A session will timeout after 30 seconds of inactivity. Activity is defined as a request or poll on the session. When long polling a pending poll will not keep a session from timing out.

The complete URL to create a new session on a machine at 10.10.20.149 would be:

http://10.10.20.149/rest/new

The returned session data is shown below.


```
{
  "_rv": 0,
  "session": "37b1ab3047ceadc4d535e7c2d0a0ce0d"
}
```

Calling functions

Functions are called by doing post requests to the /rest/request path. The complete URL for a function call to the example session above would be:

http://10.10.20.149/rest/request/37b1ab3047ceadc4d535e7c2d0a0ce0d

The JSON input required to call the Data_openFile function would be:

```
{
   "call": "Data_openFile",
   "params": {
     "filename": "my_example_file.txt",
     "mode": 0
   }
}
```

Alternately, the parameters may be supplied positionally, instead of as key-value pairs. The same call with positional parameters would be:

```
{
    "call": "Data_openFile",
    "params": ["my_example_file.txt", 0]
}
```

Output, assuming the request succeeded, would be something like:

```
{
   "_rv": 0,
   "filesize": 1337
}
```

Getting ASYNC responses and EVENTS

Before events and async responses will be returned correctly you will need to explicitly listen for them. This is done with the following call:

```
{
    "call": "listen",
    "params": {
       "events": ["SB_actorAvailable", "SB_actorExited",
       "Comm_callConnected", "Comm_callTerminated"]
    }
}
```

This call associates the name with the call. If you don't do a listen then events and responses will be unnamed, making them difficult to recognize. You can call listen multiple times and each can include multiple names if desired. Once your client is listening for an event or response there is no way to unlisten.

A GET request to the /rest/poll path or the /longpoll.rb path will return JSON structures containing event descriptions. Multiple or no events can be returned in a single request.

The complete URL for a fast poll to the example session would be:

http://10.10.20.149/rest/poll/37b1ab3047ceadc4d535e7c2d0a0ce0d

A fast poll will return data immediately. If no events are available the poll will return an empty response. Long polls allow the request to linger for a specified timeout before returning an empty response. If an event becomes available while the poll is pending then the poll completes immediately and returns the event data. When the long poll completes the client should immediately issue a new long poll so that a poll is continuously pending. This allows much lower latency in getting events without having to poll more often than necessary. The complete URL for a long poll to the example session would be:

http://10.10.20.149/longpoll.rb?session=37b1ab3047ceadc4d535e7c2d0a0ce0d&timeout=10

The timeout parameter may be omitted, in which case a 5 second timeout is assumed. The timeout should be kept short enough so that several polls will be issued during the 30 second session timeout window.

Only one poll should be issued at any given time. Issuing more than one poll at the same time will result in undefined behavior. Long polling and fast polling should not be mixed in a session.

An example response is shown below.


```
{
  "params": {
    "actor": "adminsh.adminsh_01",
    "id": 143
  },
  "call": "SB_actorAvailable"
}
{
  "params": {
    "name": "adminsh.adminsh_01",
    "id": 143
  },
  "call": "SB_actorExited"
}
```


Errors

Errors will result in a JSON structure containing the error description. If the error occurs during the processing of the GET or POST, an invalid session number, for instance, the error structure will have a _rv value of -1, and a message field that describes the error. Some errors will also return backtrace information. These are internal interpreter errors and should be reported as bugs.

An example error is shown below:


```
{
  "message": "No such session 01f586531aafc9abd673cccdab6c5850",
  "_rv": -1
}
```

SysAdmin Class Documentation

Overview

SysAdmin provides system administration and network configuration services.

Functions by Group

General capabilities

Communications port management

Managing interface Link state

IPv4 Configuration

Configuration

VLan configuration

Routing table maintenance

NTP Server configuration

DNS Configuration

Time and Date

External stimulus and logging

System services

SYNC getAesCapable()

determine if the system is AES capable or not

privilege level USER

Parameters

Outputs

_rv	*integer*	0 = Not capable, 1 = Capable

SYNC getNumberOfInterfaces()

Retrieve the number of network connections available

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Number of available network interfaces

SYNC setHostname(ConstStringZ hostname)

sets the system hostname

privilege level ADMIN

Parameters

Inputs

hostname	*string*	**UNDOCUMENTED**

Outputs

_rv	*integer*	Hostname

SYNC getHostname(StringZ hostname, size_t hostname_size)

gets the system hostname

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Hostname
hostname	*string*	**UNDOCUMENTED**

SYNC setActivePort(int ifnum)

Set the port to be used by Comm.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Port to be used for VC communications.

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getActivePort()

Get the port used for video conferencing.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Active port or -1 if no port is selected.

EVENT activePortChanged(int ifnum)

Issued when the active port changes

Parameters

ifnum	*integer*	New active port

SYNC getLinkState(int ifnum, int *state)

Gets the link state for a given interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
state	*integer*	Current Link State

NoCarrier=0	Link NoCarrier
Binding=1	Link Binding
Connected=2	Link Connected
NoAddress=3	Link NoAddress
Bonded=4	Link Bonded

EVENT linkStateChanged(int ifnum, int state)

Notification of change of link state

Parameters

ifnum	*integer*	Interface that has changed state
state	*integer*	

Down=0	Link went down
Up=1	Link came up

SYNC setLinkSpeed(int ifnum, int linkSpeed)

Configure speed of an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to configure
linkSpeed	*integer*	Speed to configure (10, 100, 1000)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLinkSpeed(int ifnum, int *linkSpeed)

Retrieve the current link speed

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
linkSpeed	*integer*	Returned speed of interface (10, 100, 1000)

EVENT linkSpeedChanged(int ifnum, int linkSpeed)

Notification of a change in link speed on an interface

Parameters

ifnum	*integer*	Interface whose speed has changed
linkSpeed	*integer*	New speed of interface

SYNC setLinkDuplex(int ifnum, int linkDuplex)

Set duplex of a link

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to change
linkDuplex	*integer*	New duplex mode of link
		Half=0 Half Duplex
		Full=1 Full Duplex

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLinkDuplex(int ifnum, int *linkDuplex)

get the current duplex mode of a link

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
linkDuplex	*integer*	Current link duplex mode
		Half=0 Half Duplex
		Full=1 Full Duplex

EVENT linkDuplexChanged(int ifnum, int linkDuplex)

Notification of link Duplex change

Parameters

ifnum	*integer*	Interface that has changed.
linkDuplex	*integer*	New duplex mode of link
		Half=0 Half Duplex
		Full=1 Full Duplex

SYNC setLinkAutoNeg(int ifnum, int linkAutoNeg)

Enable or disable autonegotiation of link mode.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to change.
linkAutoNeg	*integer*	Disabled=0 Disable autonegotiation
		Enabled=1 Enable autonegotiation

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLinkAutoNeg(int ifnum, int *linkAutoNeg)

retrieve autonegotiation capability.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query.

Outputs

_rv	*integer*	Return status (0 = success)
linkAutoNeg	*integer*	Current autonegotiation status
		Disabled=0 Disable autonegotiation
		Enabled=1 Enable autonegotiation

EVENT linkAutoNegChanged(int ifnum, int linkAutoNeg)

Notification of change in link autonegotiation enable.

Parameters

ifnum	*integer*	Interface that has changed.
linkAutoNeg	*integer*	New autonegotiation status

Disabled=0	Disable autonegotiation
Enabled=1	Enable autonegotiation

SYNC setIPv4StaticConfig(int ifnum, int dhcp, ConstStringZ address, ConstStringZ netmask, ConstStringZ broadcast)

Set IPv4 configuration of an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to configure.
dhcp	*integer*	0 = use static configuration, 1 = use DHCP for configuration
address	*string*	Static IP address (must be valid when dhcp=0, ignored when dhcp=1)
netmask	*string*	Static IP netmask (must be valid when dhcp=0, ignored when dhcp=1)
broadcast	*string*	Static broadcast address (must be valid when dhcp=0, tested when dhcp=1)

Outputs

_rv	*integer*	Return status

0	success
- EINVAL	address, netmask, or broadcast are not valid ip addresses or NULL or empty

SYNC getIPv4StaticConfig(int ifnum, int *dhcp, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Get the current IPv4 configuration for an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
dhcp	*integer*	

Static=0	Static IP configuration
DHCP=1	use DHCP configuration

address	*string*	Static IP address (empty if dhcp == 1)
netmask	*string*	Static netmask (empty if dhcp == 1)
broadcast	*string*	Static broadcast address (empty if dhcp == 1)

EVENT ipv4StaticConfigUpdated()

Notification that the IPv4 configuration changed for an interface

SYNC getIPv4Config(int ifnum, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Retrieve the active IP configuration for an interface, If the interface is configured with DHCP then this
call will return the allocated network configuration

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
address	*string*	IP address of interface
netmask	*string*	netmask of interface
broadcast	*string*	broadcast address of interface

SYNC getActiveIPv4Config(int *ifnum, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Retrieve the active IP configuration for Helium. This is bond0 if bonding is enabled, otherwise, it is the first valid ip address.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
ifnum	*integer*	of Active interface.
address	*string*	IP address of interface
netmask	*string*	netmask of interface
broadcast	*string*	broadcast address of interface

EVENT ipv4ConfigUpdated()

Notification that the active configuration of a network interface has changed

SYNC setIPv6StaticConfig(int ifnum, int enable, int autoconfig, ConstStringZ address, ConstStringZ gateway)

Set the IPv6 configuration of an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to configure	
enable	*integer*	Disable=0	disable IPv6 for this interface
		Enable=1	enable IPv6 for this interface
autoconfig	*integer*	Manual=0	use manual IPv6 address assignment
		Auto=1	use automatic IPv6 address assignment
address	*string*	manually assigned IPv6 address (ignored if automatic address configuration enabled)	
gateway	*string*	IPv6 gateway address (ignored if automatic address configuration enabled)	

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getIPv6Config(int ifnum, int *enable, int *autoconfig, StringZ address, int address_size, StringZ gateway, int gateway_size)

Retrieve the current configuration for an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to query	

Outputs

_rv	*integer*	Return status (0 = success)	
enable	*integer*	Disabled=0	IPv6 is disabled for this interface
		Enabled=1	IPv6 is enabled for this interface
autoconfig	*integer*	Manual=0	this device uses manual IPv6 address assignment
		Auto=1	this device uses automatic IPv6 address assignment
address	*string*	currently assigned IPv6 address	
gateway	*string*	current IPv6 gateway address	

EVENT ipv6StaticConfigChanged(int ifnum)

Issued when a new IPv6 configuration is set using setIPv6StaticConfig.

Parameters

ifnum	*integer*	Interface with new configuration.

EVENT ipv6ConfigChanged(int ifnum)

Issued when the active IPv6 address or gateway address changes for an interface.

Parameters

ifnum	*integer*	Interface with new active configuration.

SYNC addIPv4AliasAddress(int ifnum, ConstStringZ address, ConstStringZ netmask, ConstStringZ broadcast)

create an IP alias on an ethernet interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to add address to
address	*string*	New alias address
netmask	*string*	New alias netmask
broadcast	*string*	New alias broadcast address

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteIPv4AliasAddress(int ifnum, ConstStringZ address)

Remove an IP alias from an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to remove address from
address	*string*	Alias address to remove

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getAliasEnumerator(int ifnum)

Start an enumeration of aliases for an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to enumerate

Outputs

_rv	*integer*	Enumerator ID

SYNC enumerateNextAlias(int id, StringZ address, int address_size)

Retrieve the next alias address in an enumeration

privilege level ADMIN

Parameters

Inputs

id	*integer*	Enumerator ID from getAliasEnumerator()

Outputs

_rv	*integer*	status 0 address contains a valid alias address -1 end of enumeration
address	*string*	Returned alias address

SYNC addIPv6AliasAddress(int ifnum, ConstStringZ address)

Add an IPv6 address to an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to add address to
address	*string*	IPv6 address with prefix length in slash notation

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteIPv6AliasAddress(int ifnum, ConstStringZ address)

remove an IPv6 address from an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to remove address from
address	*string*	address to remove with prefix length

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getIPv6AliasEnumerator(int ifnum)

get an enumerator id for the IPv6 address aliases on an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to enumerate

Outputs

_rv	*integer*	enumerator id to be used in enumerateNextIPv6Alias or negative error

SYNC enumerateNextIPv6Alias(int id, StringZ address, int address_size)

Get the next address in an enumeration.

privilege level ADMIN

Parameters

Inputs

id	*integer*	enumerator id

Outputs

_rv	*integer*	status 0 buffer contains a valid IPv6 address and prefix length - end of enumeration or invalid enumerator 1
address	*string*	buffer to receive IPv6 address

SYNC setVlan(int ifnum, int vlan)

set the VLAN to be used on an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to set
vlan	*integer*	VLAN identifier 0-4094

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getVlan(int ifnum, int *vlan)

get the VLAN setting for an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to query

Outputs

_rv	*integer*	Return status (0 = success)
vlan	*integer*	VLAN identifier 0-4094

EVENT vlanChanged(int ifnum,int vlanId)

issued when the VLAN configuration of an interface is changed

Parameters

ifnum	*integer*	interface that changed
vlanId	*integer*	VLAN identifier 0-4094

SYNC setIPv4StaticGateway(ConstStringZ gateway)

Sets the IPv4 gateway to be used for static network configuration

privilege level ADMIN

Parameters

Inputs

gateway	*string*	IP address of the default gateway

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getIPv4StaticGateway(StringZ gateway, size_t gateway_size)

Retrieve the current static gateway configuration

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
gateway	*string*	IP address of default gateway

EVENT ipv4StaticGatewayUpdated()

Notification of a change in static gateway configuration

SYNC getIPv4Gateway(StringZ gateway, size_t gateway_size)

Retrieve the active IPv4 gateway

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
gateway	*string*	IP address of the current active default gateway

EVENT ipv4GatewayUpdated()

Notification of a change in default IPv4 gateway

SYNC addRoute(ConstStringZ family, ConstStringZ net, ConstStringZ netmask, ConstStringZ gateway, int metric, int mss, ConstStringZ interface)

Create a new route entry

privilege level ADMIN

Parameters

Inputs

family	*string*	address family IPv4=0 IPv4 IPv6=1 IPv6
net	*string*	address
netmask	*string*	network address mask (0 if address is a host)
gateway	*string*	gateway IP address (empty string if no gateway)
metric	*integer*	metric field (0 to omit)
mss	*integer*	max segment size for this route (0 == default)
interface	*string*	interface to be used for this route (empty string to omit)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteRoute(ConstStringZ family, ConstStringZ net,

ConstStringZ netmask)

Delete a route entry

privilege level ADMIN

Parameters

Inputs

family	*string*	address family IPv4=0 IPv4 IPv6=1 IPv6
net	*string*	address
netmask	*string*	network address mask (0 if address is a host)

Outputs

_rv	*integer*	Return status (0 = success)

EVENT routeChanged()

issued when the routing table is changed

SYNC getRouteEnumerator()

start an enumeration of the routing table

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	enumerator or negative error indication

SYNC enumerateNextRoute(int id, StringZ route, int route_size)

returns an entry in the routing table in csv format
field order is addr, mask, gateway, flags, metric, interface.

privilege level ADMIN

Parameters

Inputs

id	*integer*	enumerator id

Outputs

_rv	*integer*	Return status (0 = success)
route	*string*	buffer to receive route description

SYNC setStaticNTPServer(ConstStringZ ntpserver)

set the NTP server to use if there is no DHCP supplied server

privilege level ADMIN

Parameters

Inputs

`ntpserver`	*string*	IP address of an NTP server

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC getStaticNTPServer(StringZ ntpserver, size_t ntpserver_size)`

Retrieve the current statically configured NTP server

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`ntpserver`	*string*	IP address of the NTP server

`EVENT staticNTPServerUpdated()`

Notification of a change in static NTP server configuration

`SYNC getNTPServer(StringZ ntpserver, size_t ntpserver_size)`

Retrieve the currently active NTP server

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`ntpserver`	*string*	IP address of the NTP server

`EVENT ntpServerUpdated()`

Notification of a change in active NTP server

`SYNC setStaticDNSServers(ConstStringZ dnsServers)`

Set a list of DNS servers to use

privilege level ADMIN

Parameters

Inputs

dnsServers	*string*	Space separated list of IP addresses of DNS servers

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getStaticDNSServers(StringZ dnsServers, size_t dnsServers_size)

Retrieve the currently configured list of DNS servers

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsServers	*string*	Space separated list of DNS servers

EVENT staticDNSServersUpdated()

Notification that the configured list of DNS servers has changed

SYNC getDNSServers(StringZ dnsServers, size_t dnsServers_size)

Retrieve the active list of DNS servers

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsServers	*string*	Space separated list of DNS server IP addresses

EVENT dnsServersUpdated()

Notification that the active list of DNS servers has changed

SYNC setStaticDNSDomain(ConstStringZ dnsDomain)

Set the DNS domain to use if none is supplied by DHCP

privilege level ADMIN

Parameters

Inputs

dnsDomain	*string*	DNS domain string

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getStaticDNSDomain(StringZ dnsDomain, size_t dnsDomain_size)

Retrieve the configured static DNS domain

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsDomain	*string*	Configured DNS domain name

EVENT staticDNSDomainUpdated()

Notification that the configuration for DNS domain has changed

SYNC getDNSDomain(StringZ dnsDomain, size_t dnsDomain_size)

Retrieve the active DNS domain name

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsDomain	*string*	Active DNS domain name

EVENT dnsDomainUpdated()

Notification that the active DNS domain has changed

SYNC setStaticDNSSearch(ConstStringZ dnsSearch)

Set list of DNS domains to search

privilege level ADMIN

Parameters

Inputs

dnsSearch	*string*	Space separated list of DNS domains

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getStaticDNSSearch(StringZ dnsSearch, size_t dnsSearch_size)

Retrieve current static configuration of DNS search list

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsSearch	*string*	Space separated list of DNS domains

EVENT staticDNSSearchUpdated()

Notification of change to static DNS search list configuration

SYNC getDNSSearch(StringZ dnsSearch, size_t dnsSearch_size)

Retrieve the currently active DNS search list

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsSearch	*string*	Space separated list of DNS domains

EVENT dnsSearchUpdated()

Notification that active DNS search list has changed

SYNC setInterfaceEnable(int ifnum, int enable)

Enable or disable an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to configure	
enable	*integer*	Disabled=0	Disable interface

Enabled=1 Enable interface

Outputs

_rv *integer* Return status (0 = success)

SYNC getInterfaceEnable(int ifnum, int *enable)

Retrieve the current enable status of an interface

privilege level ADMIN

Parameters

Inputs

ifnum *integer* Interface to query

Outputs

_rv *integer* Return status (0 = success)

enable *integer* Disabled=0 Disable interface

Enabled=1 Enable interface

EVENT interfaceEnableUpdated(int ifnum)

Notification of a change in enable status for an interface

Parameters

ifnum *integer* Interface that changed

SYNC changePassword(ConstStringZ username, ConstStringZ password)

Change the password for a SOAP user

privilege level ADMIN

Parameters

Inputs

username *string* Name of user to change

password *string* New password for user

Outputs

_rv *integer* Return status (0 = success)

EVENT passwordChanged(ConstStringZ username)

Issued when a password is changed.

Parameters

username *string* user name that password was changed for

SYNC setTime(time_t utcTime)

Set the current system time

privilege level ADMIN

Parameters

Inputs

utcTime	*integer*	UTC time in seconds from the epoch

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getTime(time_t *utcTime)

Retrieve the current system time

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
utcTime	*integer*	Current system time in seconds since the epoch

EVENT timeUpdated()

Notification that system time has changed due to setTime

SYNC setTimeZone(ConstStringZ timezone)

Set the system time zone

privilege level ADMIN

Parameters

Inputs

timezone	*string*	A time zone string such as "America/Chicago"

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getTimeZone(StringZ timezone, size_t timezone_size)

Retrieve the current system time zone

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
timezone	*string*	Current system time zone string

EVENT timeZoneUpdated()

Notification that the system time zone has been changed

SYNC getTimeZoneOffset(ConstStringZ timezone, int* offset)

Retrieve the offset in seconds from GMT of the named timezone

privilege level ADMIN

Parameters

Inputs

timezone	*string*	Timezone to calculate

Outputs

_rv	*integer*	Return status (0 = success)
offset	*integer*	Offset of time zone in seconds west of GMT

ASYNC adminEvent(ConstStringZ data)

Send an external command to SysAdmin through a script

privilege level ADMIN

Parameters

Inputs

data	*string*	Command text

SYNC setLogLevel(int mask)

Set the log level for the sysadmin service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

returns the current log level mask of sysadmin

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`mask`	*integer*	returned mask

`EVENT logLevelChanged(int mask)`

issued when the sysadmin log level changes

Parameters

`mask`	*integer*	new log level mask

`EVENT maintenanceConfigUpdated( )`

Notification that the maintenence tunnel configuration has changed

`SYNC lockReset( )`

lock the system from doing a reset until lock is released

privilege level USER

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC resetIsLocked( )`

is the reset lock on

privilege level USER

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC unlockReset( )`

un lock the system from doing a reset until lock is released

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)

SYNC reboot(ConstStringZ reason)

Request a system reboot

privilege level USER

Parameters

Inputs

reason	*string*	String to be placed in the reset log

Outputs

_rv	*integer*	Return status (0 = success)

SYNC reset(ConstStringZ reason, int options)

Request a system reset with options

privilege level ADMIN

Parameters

Inputs

reason	*string*	String to be appended to reset log
options	*integer*	Option mask for this reset
		Defaults=0x0001 Reset to defaults
		Swap=0x0004 Swap boot partition

Outputs

_rv	*integer*	Return status (0 = success)

SYNC shutdown(ConstStringZ reason)

Request a system shutdown

privilege level USER

Parameters

Inputs

reason	*string*	String to be placed in the reset log

Outputs

_rv	*integer*	Return status (0 = success)

EVENT rebooting( )

Notification that a system reboot is about to occur

EVENT shuttingDown( )

Notification that a system shutdown is about to occur

SYNC getRunLevel(int* run_level)

Retrieve the current system run level

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
run_level	*integer*	Current run level (0-6)

SYNC setRunLevel(int run_level)

Request the system to transition to a new run level

privilege level ADMIN

Parameters

Inputs

run_level	*integer*	New run level for system (0-6)

Outputs

_rv	*integer*	Return status (0 = success)

EVENT runLevelUpdated( )

Notification that a new system run level has been set

SYNC setSSHServiceEnable(int enable)

Enable or disable SSH access to the system

privilege level ADMIN

Parameters

Inputs

enable	*integer*	Disabled=0	Disable SSH service
		Enabled=1	Enable SSH service

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getSSHServiceEnable(int *enable)

Retrieve the current SSH service enable status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
enable	*integer*	Returned enable status
	Disabled=0	Disable SSH service
	Enabled=1	Enable SSH service

EVENT serviceEnableSSHChanged(int enable)

Issued when the SSH service enable is changed

Parameters

enable	*integer*	
	Disabled=0	Disable interface
	Enabled=1	Enable interface

SYNC setHTTPServiceEnable(int enable)

enable or disable HTTP/SOAP access to the system

privilege level ADMIN

Parameters

Inputs

enable	*integer*	
	Disabled=0	Disable interface
	Enabled=1	Enable interface

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getHTTPServiceEnable(int *enable)

Retrieve the current enable status for web services

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
enable	*integer*	Returned enable status
	Disabled=0	Disable interface

Enabled=1 Enable interface

EVENT serviceEnableHTTPChanged(int enable)

Issued when the HTTP service is enabled or disabled

Parameters

enable	*integer*	New HTTP enable status (0 = disabled, 1 = enabled)

ASYNC startTcpdump(ConstStringZ tcpdump_filter)

invoked to start tcpdump and supply command line arguments to tcpdump program. Once started tcpdump records data into data files and rotates them based on size, each new file rotation will cause a tcpdumpDataReady response.

privilege level ADMIN

Parameters

Inputs

tcpdump_filter	*string*	allows user supplied tcpdump port filter, if null or empty string then default filter is used

ASYNC stopTcpdump( )

invoked to stop tcpdump, after tcpdump is stopped expect one tcpdumpDataReady response that provides any data in the not yet rotated tcpdump data file

privilege level ADMIN

SYNC getTcpdumpEnable(int* tcpdump_enabled)

request to request if tcpdump is started or is stoped

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
tcpdump_enabled	*integer*	0 means tcpdump is stopped, non-zero means tcpdump is started

SYNC getTcpdumpFilename(StringZ filename, size_t filename_size)

get the name of the current tcpdump file for use with Data_openFile. An empty value indicates that no tcpdump file exists.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 = success -EINVAL = filename is null or filename_size is 0 or less
filename	*string*	a buffer to hold the returned filename

SYNC getTcpdumpFilter(StringZ filter, size_t filter_size)

get the filter of the current tcpdump. If tcpdump is disabled then the last known filter is returned

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status 0 Success -EINVAL Filter is null or filter_size is 0 or less
filter	*string*	a buffer to hold the returned filter

EVENT tcpdumpEnableChanged(int tcpdump_enabled)

event / response sent when tcpdump is started or stoped

Parameters

tcpdump_enabled	*integer*	0 means tcpdump is stopped, non-zero means tcpdump is started

EVENT tcpdumpDataReady(ConstStringZ file_name, int file_size)

Notification sent when a new tcpdump file is available for collection via Data_openFile

Parameters

file_name	*string*	Tcpdump file name
file_size	*integer*	Tcpdump file size in bytes

SYNC setSyslogServer(ConstStringZ hostname)

Configures remote logging. Expect the event syslogServerUpdated if the hostname does change.

privilege level ADMIN

Parameters

Inputs

hostname	*string*	remote host that can send and receive messages to device logging daemon. Use IND name if DNS is operational or dotted quad if not. An empty value will disable remote logging.

Outputs

_rv	*integer*	Return status

0	Success
-EINVAL	Hostname not valid IP address

SYNC getSyslogServer(StringZ hostname, size_t hostname_size)

Returns current remote logging hostname.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
hostname	*string*	Current configured remote logging hostname, expect IND name, dotted quad, or empty string.

EVENT syslogServerUpdated()

Occurs when remote logging hostname is changed.

Audio Class Documentation

Overview

Functions by Group

Other Functions

ASYNC setVolume(int volLevel)

Set Audio speaker output volume level

privilege level USER

Parameters

Inputs

volLevel	*integer*	volume level (0-100)

SYNC getVolume(int *volLevel)

Get current Audio speaker output volume level

privilege level USER

Parameters

Outputs

_rv	*integer*	0 means success
volLevel	*integer*	volume level (0-100)

SYNC mute(ConstStringZ inChoice)

Mute all local inputs, works only during active call.

privilege level USER

Parameters

Inputs

inChoice	*string*	Mute choice 1) all - mute all inputs

Outputs

_rv	*integer*	0 means success, 1 means No action taken

SYNC unmute()

Reverse of mute. Unmute currently muted inputs.

privilege level USER

Parameters

Outputs

_rv	*integer*	0 means success

SYNC getMuteState(StringZ state, size_t state_size)

Get mute state. Returns "muted_mic" or "muted_all" or "unmuted"

privilege level USER

Parameters

Outputs

_rv	*integer*	0 means success
state	*string*	Mute state 1) muted_mic: muted only active michrophone, 2) muted_all: muted all inputs 3) unmuted: unmuted all inputs

SYNC setActiveSpeaker(ConstStringZ speaker)

Set active speaker selected by user for listening.

privilege level ADMIN

Parameters

Inputs

speaker	*string*	Speaker choice: 1) lineout0: Line-out speaker, 2) hdmi0: HDMI speaker, 3) dvi0: DVI speaker, 4) phone: Tethered phone speaker

Outputs

_rv	*integer*	0 means success, 1 means No action taken

SYNC getActiveSpeaker(StringZ speaker, size_t speaker_size)

Get current active speaker.

privilege level USER

Parameters

Outputs

_rv	*integer*	0 means success
speaker	*string*	1) lineout0: Line-out speaker, 2) hdmi0: HDMI speaker, 3) dvi0: DVI speaker, 4) phone: Tethered phone speaker

SYNC getActiveMic(StringZ mic, size_t mic_size)

Get current active microphone.

privilege level USER

Parameters

Outputs

_rv	*integer*	0 means success
mic	*string*	1) linein0: Line-in mic, 2) MicPod: single micpod, 3) MicPods: 2 micpods, 4) none: no active mic

SYNC setPlaybackInputs(ConstStringZ playInputs)

Set additional stereo inputs to mix for local and far-end playback. Can be more than 1 at a time by comma separated.

privilege level ADMIN

Parameters

Inputs

playInputs	*string*	Additional playback inputs to mixer: 1) hdmi0: HDMI input, 2) dvi0: DVI input, 3) linein0: Line-in, 4) none, 5) hdmi0,dvi0,linein0

Outputs

_rv	*integer*	0 means success

SYNC getPlaybackInputs(StringZ playInputs, size_t playInputs_size)

Get current playback stereo inputs being mixed.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 means success
playInputs	*string*	Additional inputs for playback: 1) hdmi0: HDMI input, 2) dvi0: DVI input, 3) linein0: Line-in, 4) none: None selected

SYNC setLogLevel(int mask)

Set Audio log level

privilege level ADMIN

Parameters

Inputs

mask	*integer*	log level

Outputs

_rv	*integer*	0 means success

SYNC getLogLevel(int *mask)

Get Audio log level

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 means success
mask	*integer*	log level

SYNC setLineInMap(ConstStringZ videoInput)

Set the mapping of line-in to a video input such as HDMI or DVI.

privilege level ADMIN

Parameters

Inputs

videoInput	*string*	line-in would be mixed with this input if it is part of setPlaybackInputs(): 1) hdmi0, 2) dvi0, 3) none

Outputs

_rv	*integer*	0 means success

SYNC getLineInMap(StringZ videoInput, size_t videoInput_size)

Get video input device mapped with line-in.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 means success
videoInput	*string*	video input: 1) hdmi0: HDMI input, 2) dvi0: DVI input, 3) none: None selected

SYNC setAnalogMicGain(ConstStringZ gain)

Set analog line-in mic gain.

privilege level ADMIN

Parameters

Inputs

gain	*string*	options: 1) low: Line Level (default), 2) medium: Microphone Level , 3) high: Microphone level with boost

Outputs

_rv	*integer*	0 means success

SYNC getAnalogMicGain(StringZ gain, size_t gain_size)

Get current analog line-in mic gain.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 means success
gain	*string*	options: 1) low: Line Level (default), 2) medium: Microphone Level , 3) high: Microphone level with boost

EVENT micBeamChanged(int deviceId, int beamId)

Issued whenever the mic beam selection is changed. startBeamPublish() should be invoked to start receiving this event.

Parameters

deviceId	*integer*	device ID. 0 - Mustang, 1 - Condor0, 2 - Condor1
beamId	*integer*	beam ID, one beam for each Mic. 0 - 3 for Mustang & 0 - 2 for Condors

EVENT aecDelayResult(int delay)

Issued when we detected a new delay result

Parameters

delay	*integer*	Delay value detected.

EVENT jitterDidChange(const int *call_ids, int call_ids_count, const int *jitter, int jitter_count)

Issued when jitter value of any channel is updated

Parameters

call_ids	*array of integer*	call IDs
jitter	*array of integer*	Jitter value buffer pointer

EVENT delayEstimate(int call_id, int channel_id, int delay, int channel_flag)

Issued when the algorithmic delay of an audio channel is updated

Parameters

call_id	*integer*	call ID
channel_id	*integer*	channel ID
delay	*integer*	new audio delay estimate
channel_flag	*integer*	Channel flag 0-Tx(encoder), 1-Rx(decoder)

EVENT playSoundDidEnd(int chan_id, int repeat_instance, int status)

Issued when playback is over

Parameters

chan_id	*integer*	Play channel ID
repeat_instance	*integer*	Play Repeat instance
status	*integer*	0 = success, -1 = unknown error, -16 = file read error

EVENT volumeChanged(int volLevel)

Issued when the speaker volume is updated

Parameters

volLevel	*integer*	volume level (0-100)

EVENT muted(int status)

Issued when the mute state of the system is updated

Parameters

status	*integer*	1-muted/0-unmuted

EVENT logLevelChanged(int mask)

Issued when the Audio log level is updated

Parameters

mask	*integer*	New log level

EVENT emptyWaitQ(int ch_id)

empty wait queue of a channel

Parameters

ch_id	*integer*	channel ID

EVENT emptyToneQ(int ch_id)

empty tone wait queue of a tone channel

Parameters

ch_id	*integer*	channel ID

EVENT activeMicChanged(ConstStringZ mic)

Issued when the active microphone is updated.

Parameters

mic	*string*	1) linein0 - Line-in mic, 2) lifelink - lifelink device 3) none - none is selected as active mic

EVENT recordingStatusChanged(ConstStringZ status)

Issued when the recording status has changed

Parameters

`status`	*string*	On / Off

`EVENT recordingAvailable(ConstStringZ status)`

Issued when the recording status has changed

Parameters

`status`	*string*	should set to True when it's available, False when it is not.

`EVENT activeSpeakerChanged(ConstStringZ speaker)`

Issued when the active speaker is updated.

Parameters

`speaker`	*string*	1) lineout0: Line-out speaker, 2) hdmi0: HDMI speaker, 3) dvi0: DVI speaker, 4) phone: Tethered phone speaker

`EVENT playbackInputsChanged(ConstStringZ playInputs)`

Issued when the playback inputs are updated.

Parameters

`playInputs`	*string*	New playback inputs: 1) hdmi0: HDMI input, 2) dvi0: DVI input, 3) linein0: Line-in, 4) none: None selected

`EVENT lineInMapChanged(ConstStringZ videoInput)`

Issued when line-in mapping is updated.

Parameters

`videoInput`	*string*	video input: 1) hdmi0: HDMI input, 2) dvi0: DVI input, 3) none: None selected

`EVENT analogMicGainChanged(ConstStringZ gain)`

Issued when the analog line-in mic gain is updated.

Parameters

`gain`	*string*	options: 1) low: Line Level (default), 2) medium: Microphone Level , 3) high: Microphone level with boost

CDR Class Documentation

Overview

Functions by Group

Other Functions

SYNC setMaxNumberOfFiles(int fileCount)

This API is used to set the maximum number of files to be used for cdr

privilege level ADMIN

Parameters

Inputs

fileCount	*integer*	This indicates that how many files can be create for CDR

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getMaxNumberOfFiles(int *fileCount)

This is used to obtain the currently set maximum number of files

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
fileCount	*integer*	This indicates that how many files can be create

for CDR

EVENT maxNumberOfFilesUpdated()

This is EVENT throws when maximum number of files changed

SYNC setMaxCdrNodes(int maxCdrNodes)

Set the Max CDR Nodes value. NOTE: This will erase existing cdr records

privilege level ADMIN

Parameters

Inputs

maxCdrNodes	*integer*	Max CDR records

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getMaxCdrNodes(int *maxCdrNodes)

Get the max CDR nodes value

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
maxCdrNodes	*integer*	returns the max no: of CDR nodes

EVENT maxCdrNodesUpdated()

Notification of Max CDR Nodes update status

SYNC getFileLocation(StringZ filePath, int filePath_size)

This is used to obtain the currently set CDR files location

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
filePath	*string*	This indicates the file location for the cdr files

ASYNC getAllCdrs()

RPC function to get the CDRs into an iterator. This function will concatenate all CDRs to /tmp/cdr.xml and adds the xml declaration and sequoia header node

privilege level ADMIN

RESPONSE getAllCdrsDone(int iterator, size_t sizeInBytes)

This response gets when do getAllCdrs

Parameters

iterator	*integer*	This indicates the interator number
sizeInBytes	*unsigned int*	This indicates the file size

SYNC getNextChunk(int iterator, StringZ buffer, size_t buffer_size)

This is used to obtain the next chunk of the cdr

privilege level ADMIN

Parameters

Inputs

iterator	*integer*	This indicates the interator number

Outputs

_rv	*integer*	Return status (0 = success)
buffer	*string*	This returns the buffered data for the cdr. It returns only MaxFileSize in terms of bytes

SYNC setMaxFileSize(int fileSize)

This is used to set the maximum file size in terms of bytes

privilege level ADMIN

Parameters

Inputs

fileSize	*integer*	This indicates the file size to be returned when we invoke getNextChunk

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getMaxFileSize(int *fileSize)

This is used to invoke the current maximum file size set

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`fileSize`	*integer*	This indicates the file size data to be returned when we invoke getNextChunk

`EVENT maxFileSizeUpdated()`

This event throws when MaxFileSize updated

Camera Class Documentation

Overview

Provides control and configuration services for camera input devices.

Functions by Group

General camera status

getConnected
connectedChanged
getSupported
setSleepEnabled

Anti-Flicker settings

setAntiFlicker
getAntiFlicker
antiFlickerChanged

Exposure settings

setAutoExposureMethod
getAutoExposureMethod
autoExposureMethodChanged
setBrightness
getBrightness
brightnessChanged

Auto-Focus settings

setAutoFocusEnabled
getAutoFocusEnabled
autoFocusEnabledChanged

White-Balance settings

setColorCorrection
getColorCorrection
colorCorrectionChanged
setGrGbOffset
getGrGbOffset
grGbOffsetChanged

Pan/Tilt/Zoom control

setLockEnabled
getLockEnabled
lockEnabledChanged
setPreset
setPresetPosition
getPresetPosition
recallPreset

Firmware upgrade settings and status

Video output settings

Led control

VISCA control configuration

SYNC getConnected(lsdevhandle_t dev, int *connected)

Determine if the camera is currently connected.

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
connected	*integer*	Returned connected state (0 == disconnected, 1 == connected)

EVENT connectedChanged(lsdevhandle_t dev, int connected)

Notification of change in camera connected state

Parameters

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
connected	*integer*	New connected state (0 == disconnected, 1 == connected)

SYNC getSupported(lsdevhandle_t dev, int *supported)

Determine if a camera is supported on the specified device interface (does not indicate if a camera is currently connected)

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
supported	*integer*	Returned supported state (0 == not-supported, 1 == supported)

ASYNC setSleepEnabled(lsdevhandle_t dev, int enabled)

Enable or disable camera sleep mode

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

SYNC setAntiFlicker(lsdevhandle_t dev, int value)

Set the anti-flicker correction mode

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
value	*integer*	New anti-flicker mode
		50Hz=0 50Hz
		60Hz=1 60Hz
		auto=2 auto

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getAntiFlicker(lsdevhandle_t dev, int *value)

Determine the current anti-flicker correction mode

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
value	*integer*	Returned anti-flicker mode
		50Hz=0 50Hz
		60Hz=1 60Hz
		auto=2 auto

EVENT antiFlickerChanged(lsdevhandle_t dev, int value)

Notification of change of anti-flicker mode

Parameters

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
value	*integer*	New anti-flicker mode
		50Hz=0 50Hz
		60Hz=1 60Hz
		auto=2 auto

SYNC setAutoExposureMethod(lsdevhandle_t dev, int value)

Set auto-exposure method

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
value	*integer*	New auto-exposure method
		manual=0 manual
		spot=1 spot
		full-frame=2 full-frame
		center-weighted=3 center-weighted

Outputs

_rv	*integer*	Return status (0 = success)

`SYNC getAutoExposureMethod(lsdevhandle_t dev, int *value)`

Get current auto-exposure method

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
value	*integer*	Returned auto-exposure method
		manual=0 manual
		spot=1 spot
		full-frame=2 full-frame
		center-weighted=3 center-weighted

`EVENT autoExposureMethodChanged(lsdevhandle_t dev, int value)`

Notification of change of auto-exposure method

Parameters

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

value	*integer*	New auto-exposure method
		manual=0 manual
		spot=1 spot
		full-frame=2 full-frame
		center-weighted=3 center-weighted

SYNC setBrightness(lsdevhandle_t dev, int value)

Set image brightness offset

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
value	*integer*	New brightness value (min = -30, max = 30)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getBrightness(lsdevhandle_t dev, int *value)

Get current brightness offset

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
value	*integer*	Returned brightness value (min = -30, max = 30)

EVENT brightnessChanged(lsdevhandle_t dev, int value)

Notification of change of brightness offset

Parameters

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
value	*integer*	New brightness value (min = -30, max = 30)

SYNC setAutoFocusEnabled(lsdevhandle_t dev, int enabled)

Enable or disable auto-focus

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getAutoFocusEnabled(lsdevhandle_t dev, int *enabled)

Determine if auto-focus is enabled

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
enabled	*integer*	Returned state (0 == disabled, 1 == enabled)

EVENT autoFocusEnabledChanged(lsdevhandle_t dev, int enabled)

Notification event issued when auto-focus enable is changed

Parameters

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

SYNC setColorCorrection(lsdevhandle_t dev, int value)

Set the desired color-correction mode

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
value	*integer*	New color-correction value auto=0 auto white-fluorescent=2 white-fluorescent cool-white-fluorescent=3 cool-white-fluorescent outdoor=4 outdoor tl84-fluorescent=5 tl84-fluorescent a28k-incandescent=6 a28k-incandescent a32k-incandescent=7 a32k-incandescent daylight-fluorescent=8 daylight-fluorescent brightline-5600k=9 brightline-5600k brightline-3200k=10 brightline-3200k

Outputs

_rv	*integer*	Return status (0 = success)

`SYNC getColorCorrection(lsdevhandle_t dev, int *value)`

Determine the current color-correction mode

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
value	*integer*	Returned color-correction value auto=0 auto white-fluorescent=2 white-fluorescent cool-white-fluorescent=3 cool-white-fluorescent outdoor=4 outdoor tl84-fluorescent=5 tl84-fluorescent a28k-incandescent=6 a28k-incandescent a32k-incandescent=7 a32k-incandescent

daylight-fluorescent=8	daylight-fluorescent
brightline-5600k=9	brightline-5600k
brightline-3200k=10	brightline-3200k

EVENT colorCorrectionChanged(lsdevhandle_t dev, int value)

Notification of change of color-correction mode

Parameters

dev *unsigned int* Camera device interface

dvi0=0x00100000	DVI camera 0
hdmi0=0x00020000	HDMI camera 0

value *integer* New color-correction value

auto=0	auto
white-fluorescent=2	white-fluorescent
cool-white-fluorescent=3	cool-white-fluorescent
outdoor=4	outdoor
tl84-fluorescent=5	tl84-fluorescent
a28k-incandescent=6	a28k-incandescent
a32k-incandescent=7	a32k-incandescent
daylight-fluorescent=8	daylight-fluorescent
brightline-5600k=9	brightline-5600k
brightline-3200k=10	brightline-3200k

SYNC setGrGbOffset(lsdevhandle_t dev, float value)

Set the GrGb gain offset (not valid when color-correction mode is set to auto)

privilege level ADMIN

Parameters

Inputs

dev *unsigned int* Camera device interface

dvi0=0x00100000	DVI camera 0
hdmi0=0x00020000	HDMI camera 0

value *float* New GrGb offset (-1.0 to 1.0)

Outputs

_rv *integer* Return status (0 = success)

SYNC getGrGbOffset(lsdevhandle_t dev, float *value)

Determine the current GrGb gain offset (not valid when color-correction mode is set to auto)

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
value	*float*	Returned GrGb offset (-1.0 to 1.0)

EVENT grGbOffsetChanged(lsdevhandle_t dev, float value)

Notification of change of GrGb gain offset

Parameters

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
value	*float*	New GrGb offset (-1.0 to 1.0)

SYNC setLockEnabled(lsdevhandle_t dev, int enabled)

Enable or disable pan/tilt/zoom lock

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == unlocked, 1 == locked)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLockEnabled(lsdevhandle_t dev, int *enabled)

Determine if pan/tilt/zoom lock is enabled

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface

dvi0=0x00100000 DVI camera 0
hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
enabled	*integer*	Returned state (0 == unlocked, 1 == locked)

EVENT lockEnabledChanged(lsdevhandle_t dev, int enabled)

Notification event issued when pan/tilt/zoom lock enable is changed

Parameters

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == unlocked, 1 == locked)

SYNC setPreset(lsdevhandle_t dev, int preset)

Save a new preset for a particular camera using the current position

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
preset	*integer*	Preset number (1 through 19)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC setPresetPosition(lsdevhandle_t dev, int preset, float pan, float tilt, float zoom)

Save a new preset for a particular camera and position

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
preset	*integer*	Preset number (1 through 19)
pan	*float*	Pan position in degrees (-180.0 to 180.0)

tilt	*float*	Tilt position in degrees (-90.0 to 90.0)
zoom	*float*	Zoom position in percent (0.0 to 100.0)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getPresetPosition(int preset, lsdevhandle_t *dev, float *pan, float *tilt, float *zoom)

Retrieve the camera device interface, pan, tilt, and zoom values for a particular stored preset

privilege level ADMIN

Parameters

Inputs

preset	*integer*	Preset number (0 through 19, 0 = home position)

Outputs

_rv	*integer*	Return status (0 = success)
dev	*unsigned int*	Returned Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
pan	*float*	Returned pan position in degrees (-180.0 to 180.0)
tilt	*float*	Returned tilt position in degrees (-90.0 to 90.0)
zoom	*float*	Returned zoom position in percent (0.0 to 100.0)

SYNC recallPreset(int preset)

Recall a previously stored preset

privilege level ADMIN

Parameters

Inputs

preset	*integer*	Preset number (0 through 19, 0 = home position for currently active camera)

Outputs

_rv	*integer*	Return status (0 = success)

ASYNC stop(lsdevhandle_t dev)

Stop all motors

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

SYNC setPosition(lsdevhandle_t dev, float pan, float tilt, float zoom)

Move the camera to a specific position

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
pan	*float*	Pan position in degrees (-180.0 to 180.0)
tilt	*float*	Tilt position in degrees (-90.0 to 90.0)
zoom	*float*	Zoom position in percent (0.0 to 100.0)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getPosition(lsdevhandle_t dev, float *pan, float *tilt, float *zoom)

Get the current camera position

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
pan	*float*	Pan position in degrees (-180.0 to 180.0)
tilt	*float*	Tilt position in degrees (-90.0 to 90.0)
zoom	*float*	Zoom position in percent (0.0 to 100.0)

ASYNC pan(lsdevhandle_t dev, int direction, float speed)

Start panning the camera

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
direction	*integer*	Direction to pan the camera negative=-1 negative direction positive=1 positive direction
speed	*float*	Movement speed -1.0 automatic speed (based on current zoom magnification) (0.0:100.0] percentage of max-supported speed

`ASYNC tilt(lsdevhandle_t dev, int direction, float speed)`

Start tilting the camera

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
direction	*integer*	Direction to tilt the camera down=-1 negative/down direction up=1 positive/up direction
speed	*float*	Movement speed -1.0 automatic speed (based on current zoom magnification) (0.0:100.0] percentage of max-supported speed

`ASYNC zoom(lsdevhandle_t dev, int direction, float speed)`

Start zooming the camera

privilege level ADMIN

Parameters

Inputs

dev	*unsigned*	Camera device interface

int	dvi0=0x00100000	DVI camera 0
	hdmi0=0x00020000	HDMI camera 0

`direction` *integer* — Direction to zoom the camera lens

wide=-1	negative/out/wide direction
tele=1	positive/in/tele direction

`speed` *float* — Movement speed

-1.0	automatic speed
(0.0:100.0]	percentage of max-supported speed

`ASYNC panNudge(lsdevhandle_t dev, int count)`

Shift the pan position by a small amount

privilege level ADMIN

Parameters

Inputs

`dev` *unsigned int* — Camera device interface

dvi0=0x00100000	DVI camera 0
hdmi0=0x00020000	HDMI camera 0

`count` *integer* — Multiple of minimum nudge distance based on current zoom magnification (negative == positive direction, positive == positive direction)

`ASYNC tiltNudge(lsdevhandle_t dev, int count)`

Shift the tilt position by a small amount

privilege level ADMIN

Parameters

Inputs

`dev` *unsigned int* — Camera device interface

dvi0=0x00100000	DVI camera 0
hdmi0=0x00020000	HDMI camera 0

`count` *integer* — Multiple of minimum nudge distance based on current zoom magnification (negative == negative/down direction, positive == positive/up direction)

`ASYNC zoomNudge(lsdevhandle_t dev, int count)`

Shift the zoom position by a small amount

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
count	*integer*	Multiple of minimum nudge distance (negative == negative/out/wide direction, positive == positive/in/tele direction)

SYNC setBackgroundUpgradeEnabled(lsdevhandle_t dev, int enabled)

Enable or disable automatic camera upgrades in the background

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getBackgroundUpgradeEnabled(lsdevhandle_t dev, int *enabled)

Determine if background camera upgrade is enabled

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
enabled	*integer*	Returned state (0 == disabled, 1 == enabled)

EVENT backgroundUpgradeEnabledChanged(lsdevhandle_t dev, int enabled)

Notification event issued when background camera upgrade enable is changed

Parameters

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

SYNC setVerticalFlipEnabled(lsdevhandle_t dev, int enabled)

Enable or disable flipping video vertically

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getVerticalFlipEnabled(lsdevhandle_t dev, int *enabled)

Determine if flipping video vertically is enabled

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
enabled	*integer*	Returned state (0 == disabled, 1 == enabled)

EVENT verticalFlipEnabledChanged(lsdevhandle_t dev, int enabled)

Notification event issued when flipping video vertically enable is changed

Parameters

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0

		hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

SYNC setHorizontalFlipEnabled(lsdevhandle_t dev, int enabled)

Enable or disable flipping video horizontally

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getHorizontalFlipEnabled(lsdevhandle_t dev, int *enabled)

Determine if flipping video horizontally is enabled

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)
enabled	*integer*	Returned state (0 == disabled, 1 == enabled)

EVENT horizontalFlipEnabledChanged(lsdevhandle_t dev, int enabled)

Notification event issued when flipping video horizontally enable is changed

Parameters

dev	*unsigned int*	Camera device interface
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0
enabled	*integer*	New state (0 == disabled, 1 == enabled)

ASYNC ledBlink(lsdevhandle_t dev, int which, int rate)

Start blinking the led at a particular rate

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
which	*integer*	Which LED to control (0 == blue, 1 == red)
rate	*integer*	Blink rate (0 to 100, 0 = slowest, 100 = fastest)

ASYNC ledBrightness(lsdevhandle_t dev, int which, int brightness)

Turn on the LED to a particular brightness

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface dvi0=0x00100000 DVI camera 0 hdmi0=0x00020000 HDMI camera 0
which	*integer*	Which LED to control (0 == blue, 1 == red)
brightness	*integer*	LED brightness (0 to 100, 0 = off, 100 = bright)

SYNC setViscaChain(lsdevhandle_t serialDevice, const lsdevhandle_t *videoInputs, int videoInputs_count)

Set the serial control VISCA chain

privilege level ADMIN

Parameters

Inputs

serialDevice	*unsigned int*	The serial device VISCA daisy chain to configure usb0=0x20000000 Physical usb port 0 usb1=0x20000001 Physical usb port 1
videoInputs	*array of unsigned*	The daisy chain of VISCA camera device interfaces controlled by serialDevice

int	dvi0=0x00100000	DVI camera 0
	hdmi0=0x00020000	HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getViscaChain(lsdevhandle_t serialDevice, lsdevhandle_t *videoInputs, int *videoInputs_count)

Get the serial control VISCA chain

privilege level ADMIN

Parameters

Inputs

serialDevice	*unsigned int*	The serial device VISCA daisy chain configuration to retrieve
		usb0=0x20000000 Physical usb port 0
		usb1=0x20000001 Physical usb port 1
videoInputs_count	*unsigned int*	max number of videoInputs to return

Outputs

_rv	*integer*	Return status (0 = success)
videoInputs	*array of unsigned int*	Returned daisy chain of VISCA camera device interfaces controlled by serialDevice
		dvi0=0x00100000 DVI camera 0
		hdmi0=0x00020000 HDMI camera 0

EVENT viscaChainChanged(lsdevhandle_t serialDevice, const lsdevhandle_t *videoInputs, int videoInputs_count)

Notification that the VISCA chain has changed

Parameters

serialDevice	*unsigned int*	The serial device VISCA daisy chain to control (DEV_IO_USB_SERIAL0, DEV_IO_USB_SERIAL1)
		usb0=0x20000000 Physical usb port 0
		usb1=0x20000001 Physical usb port 1
videoInputs	*array of unsigned int*	The daisy chain of VISCA camera device interfaces controlled by serialDevice
		dvi0=0x00100000 DVI camera 0

hdmi0=0x00020000	HDMI camera 0	

SYNC setViscaProductTypeOverride(lsdevhandle_t dev, int value)

Set the VISCA product override (force the product-type to a particular value, regardless of what is detected)

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface	
		dvi0=0x00100000	DVI camera 0
		hdmi0=0x00020000	HDMI camera 0
value	*integer*	Product override value	
		none=0	disabled
		vaddiocamhd20=13	Vaddio ClearVIEW HD-20

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getViscaProductTypeOverride(lsdevhandle_t dev, int *value)

Get the VISCA product override value

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Camera device interface	
		dvi0=0x00100000	DVI camera 0
		hdmi0=0x00020000	HDMI camera 0

Outputs

_rv	*integer*	Return status (0 = success)	
value	*integer*	Returned product override value	
		none=0	disabled
		vaddiocamhd20=13	Vaddio ClearVIEW HD-20

Comm Class Documentation

Overview

Functions by Group

H323GKConfiguration - These list the APIs that can be used to H.323 GK configurations

- setH323Details
- getH323Details
- h323DetailsUpdated
- h323Register
- getH323RegisterStatus
- h323RegisterStatusUpdated

SIPConfiguration - These list the APIs that can be used to SIP configurations

- setSipDetails
- getSipDetails
- sipDetailsUpdated
- sipRegister
- getSipRegisterStatus
- sipRegisterStatusUpdated

LSTransitConfiguration - These are the events/messages related to LS Transit

- setLSTransitDetails
- getLSTransitDetails
- lsTransitDetailsUpdated
- getLSTransitStatus
- lsTransitStatusUpdated

ConnectionsConfiguration

- ConnectionsEnableUpdated
- registerToConnectionsTriggered
- connectionsStatusUpdated
- refreshContactList
- refreshDirectoryEntries

CallManagement - These list the APIs that can be used to manage the calls.

- incomingCall
- callRinging
- callConnected
- callTransferred
- changeCallCharacteristics
- mediaChannelDisconnected
- mediaChannelConnected
- mediaChannelChanged
- terminatingCall
- remoteCallDisconnected
- remoteCallId

DTMF - These are the events/messages used for DTMF sending/receipt

- dtmfRecvd

Presentation - These are the events/messages related Presentation

- remotePresentationStarted
- remotePresentationStopped
- presentationStateChanged
- farAudioMuteStatusChanged

FECCCapability - These are the events/messages related to FECC Capability

- feccCapabilityRecvd
- feccPresetCmdRecvd
- feccMsgRecvd
- feccSetPresetCmdRecvd

CallStats

- statsEvent

GeneralConfiguration - These list of the APIs that can be used to General Configurations

- setDualVideoFeature
- getDualVideoFeature
- dualVideoFeatureUpdated
- setFarControlPropertyOverLocalCamera
- getFarControlPropertyOverLocalCamera
- farControlPropertyOnLocalCameraUpdated
- setVideoMtuCfg
- getVideoMtuCfg
- videoMtuCfgUpdated
- setQos
- getQos
- getQosExt
- qosUpdated
- setSipSecurityProperty
- getSipSecurityProperty
- sipSecurityPropertyModified
- setH323SecurityProperty
- getH323SecurityProperty
- h323SecurityPropertyModified
- setAutoBwFeature
- getAutoBwFeature
- autoBwFeatureUpdated
- setStaticNATDetails
- getStaticNATDetails
- staticNATDetailsUpdated
- setAMCFeature
- getAMCFeature
- amcFeatureUpdated
- setMaxBitrate
- getMaxBitrate
- maxBitrateUpdated
- setBWSlider
- getBWSlider
- bwSliderUpdated
- setAutoCallBitRate
- getAutoCallBitRate
- setLogLevel
- getLogLevel
- logLevelChanged

VCRecordingConfiguration - These are the events/messages related to Video Centre recording.

- setStreamingRecordingDetails
- getStreamingRecordingDetails
- streamingRecordingDetailsModified
- recordingPossible
- getRecordingPossible
- startRecording
- startRecordingDone
- stopRecording
- stopRecordingDone
- getRecordingState
- recordingStateChanged

SystemIdentification - These are the events/messages related to the system identification

- setSystemIdentification
- getSystemIdentification
- systemIdentificationUpdated

NetworkPage - These apis are related to network page options.

- setReservedPortRange
- getReservedPortRange
- portRangeUpdated

GeneralSupportPage

- customCommandExecuted
- webProxyDetailsUpdated
- preferenceForProtocolUpdated
- rvLogEnableChanged
- RFC1918VerificationStatusUpdated
- InbandDtmfEnableUpdated
- bsModeSiren14Updated
- allow2ndAudioCallUpdated
- h264PM1EnableUpdated

SIPGenSupportPage

- sipGenSupportValuesUpdated

SipSupportPage

sipSupportValuesUpdated

LSCSupportPage

ConnectionsServerURLUpdated
getConnectionsDetailsDone
mediaEncryptionForConnectionsUpdated
signalingModeForConnectionsUpdated

H323SupportPage

h323SupportValuesUpdated

SupportPage

setupTransportAddrStatusUpdated
symmetricRtpUpdated
mediaDisconnectTimerUpdated
noMediaDisconnectUpdated
H323TcpKeepaliveUpdated
H241MaxStaticMbpsCfgUpdated

SYNC setH323Details(const H323Details *pH323Details)

This is used to set the gatekeeper details

privilege level ADMIN

Parameters

Inputs

pH323Details	*structure*	STRUCTURE This is a struct containing the GK details
	bIsH323Enabled	This field controls whether or not H323 is enabled disabled=0 H323 is NOT enabled enabled=1 H323 is enabled
	strH323Name	NULL terminated string indicating the H323 Name
	strH323Extn	NULL terminated string indicating the H323 Extension
	eGkMode	The various GK Modes we support GK_MODE_OFF=0 GkMode is GK_MODE_OFF GK_MODE_AUTO=1 GkMode is GK_MODE_AUTO GK_MODE_MANUAL=2 GkMode is GK_MODE_MANUAL GK_MODE_MANUAL_H460=3 GkMode is GK_MODE_MANUAL_H460
	h323GkModeAuto	This struct needs to be filled only in case the eGkMode is having value GK_MODE_AUTO
	h323GkModeAuto.strGkIdentifier	NULL terminated string indicating the GK Identifier, applicable when GkMode is GK_MODE_AUTO
	h323GkModeManual	This struct needs to be filled only in case the eGkMode is having value GK_MODE_MANUAL
	h323GkModeManual.strGkIpAddress	NULL terminated string indicating the GK IPAddress, applicable when GkMode is GK_MODE_MANUAL
	h323GkModeManual.gkPort	positive non-zero integer, indicating the port of the GK, applicable when GkMode is GK_MODE_MANUAL
	h323GkModeH460	This struct needs to be filled only in case the eGkMode is having value GK_MODE_MANUAL_H460
	h323GkModeH460.strGkIpAddress	NULL terminated string indicating the GK IPAddress, applicable when GkMode is GK_MODE_MANUAL_H460
	h323GkModeH460.gkPort	positive non-zero integer, indicating the port of the GK, applicable when GkMode is GK_MODE_MANUAL_H460
	bIsAuthEnabled	This controls whether or not we want to do GK Authentication, this cannot have a value of 1, if GkMode is GK_MODE_OFF disabled=0 GK Authentication is NOT enabled enabled=1 GK Authentication is enabled
	strGkAuthUsername	NULL terminated string indicating the GK Auth Name, applicable only when bIsAuthEnabled=1
	strGkAuthPassword	NULL terminated string indicating the GK Auth Password, applicable only when bIsAuthEnabled=1

Outputs

_rv	*integer*	commReturnVal
	COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
	COMMRPC_FAILURE=-601	COMMRPC_FAILURE
	COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM

COMMRPC_FAILURE_NOTALLOWED_TRANSIT_IS_ENABLED=-605	COMMRPC_FAILURE_NOTALLOWED_TRANSIT_IS_ENABLED
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getH323Details(H323Details *pH323Details)

This is used to obtain the currently set GateKeeper details

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pH323Details	*structure*	STRUCTURE This is a struct containing the GK details
bIsH323Enabled		This field controls whether or not H323 is enabled disabled=0 H323 is NOT enabled enabled=1 H323 is enabled
strH323Name		NULL terminated string indicating the H323 Name
strH323Extn		NULL terminated string indicating the H323 Extension
eGkMode		The various GK Modes we support GK_MODE_OFF=0 GkMode is GK_MODE_OFF GK_MODE_AUTO=1 GkMode is GK_MODE_AUTO GK_MODE_MANUAL=2 GkMode is GK_MODE_MANUAL GK_MODE_MANUAL_H460=3 GkMode is GK_MODE_MANUAL_H460
h323GkModeAuto		This struct needs to be filled only in case the eGkMode is having value GK_MODE_AUTO
h323GkModeAuto.strGkIdentifier		NULL terminated string indicating the GK Identifier, applicable when GkMode is GK_MODE_AUTO
h323GkModeManual		This struct needs to be filled only in case the eGkMode is having value GK_MODE_MANUAL
h323GkModeManual.strGkIpAddress		NULL terminated string indicating the GK IPAddress, applicable when GkMode is GK_MODE_MANUAL
h323GkModeManual.gkPort		positive non-zero integer, indicating the port of the GK, applicable when GkMode is GK_MODE_MANUAL
h323GkModeH460		This struct needs to be filled only in case the eGkMode is having value GK_MODE_MANUAL_H460
h323GkModeH460.strGkIpAddress		NULL terminated string indicating the GK IPAddress, applicable when GkMode is GK_MODE_MANUAL_H460
h323GkModeH460.gkPort		positive non-zero integer, indicating the port of the GK, applicable when GkMode is GK_MODE_MANUAL_H460
bIsAuthEnabled		This controls whether or not we want to do GK Authentication, this cannot have a value of 1, if GkMode is GK_MODE_OFF disabled=0 GK Authentication is NOT enabled enabled=1 GK Authentication is enabled
strGkAuthUsername		NULL terminated string indicating the GK Auth Name, applicable only when bIsAuthEnabled=1
strGkAuthPassword		NULL terminated string indicating the GK Auth Password, applicable only when bIsAuthEnabled=1

EVENT h323DetailsUpdated(int bIsH323Enabled)

This is an event thrown to indicate that the GateKeeper details have been updated

Parameters

bIsH323Enabled	*integer*	This field controls whether or not H323 is enabled. disabled=0 H323 is NOT enabled enabled=1 H323 is enabled

SYNC h323Register()

This is the method exposed by comm using which anyone can force a GateKeeper registration

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getH323RegisterStatus(RegistrationStatusInfo *psStatus)

This is a method exposed using which anyone can obtain the current GK registration status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
psStatus	*structure*	STRUCTURE This struct has an enum that contains the current GK registration status and also the GK IP and Port to which the registration is being attempted
sGkAddr		This stores the IP addr and port of the GK that we are trying to register to.
eRegStatus		REG_STATUS_REGISTERED=0 REG_STATUS_REGISTERED REG_STATUS_REGISTERING=1 REG_STATUS_REGISTERING REG_STATUS_UNREGISTERED=2 REG_STATUS_UNREGISTERED REG_STATUS_UNREACHABLE=3 REG_STATUS_UNREACHABLE REG_STATUS_FAILED=4 REG_STATUS_FAILED

EVENT h323RegisterStatusUpdated(RegistrationStatusInfo * psStatus)

This is the event that is thrown by comm to indicate that the GK registration status has changed

Parameters

psStatus	*structure*	STRUCTURE This struct has an enum that contains the current GK registration status and also the GK IP and Port to which the registration is being attempted
sGkAddr		This stores the IP addr and port of the GK that we are trying to register to.
eRegStatus		REG_STATUS_REGISTERED=0 REG_STATUS_REGISTERED REG_STATUS_REGISTERING=1 REG_STATUS_REGISTERING REG_STATUS_UNREGISTERED=2 REG_STATUS_UNREGISTERED REG_STATUS_UNREACHABLE=3 REG_STATUS_UNREACHABLE REG_STATUS_FAILED=4 REG_STATUS_FAILED

SYNC setSipDetails(const SipDetails *pSipDetails,SipContextType eContextType)

This is used to set the SIP details for SIP configuration

privilege level ADMIN

Parameters

Inputs

pSipDetails	*structure*	This structure contains the actual SIP details
bIsSipEnabled		disabled=0 SIP is NOT enabled enabled=1 SIP is enabled
userName		NULL terminated string representing the UserName
authName		NULL terminated string representing the authName
authPassword		NULL terminated string representing the authPassword
eSipServerType		SIP_SERVER_AUTO=0 SIP_SERVER_AUTO SIP_SERVER_OCS=1 SIP_SERVER_OCS SIP_SERVER_OCS_MANUAL=2 SIP_SERVER_OCS_MANUAL
sStructAuto		This struct needs to be filled up only in case the eSipServerType is having value of SIP_SERVER_AUTO
sStructAuto.bIsRegistrarEnabled		This refers that SIP Registrar is disabled or enabled disabled=0 Registrar is disabled enabled=1 Registrar is enabled
sStructAuto.registrarHostName		NULL terminated string representing the SIP Registrar Hostname
sStructAuto.bIsProxyEnabled		This refers that SIP Proxy is disabled or enabled disabled=0 Proxy is disabled enabled=1 Proxy is enabled
sStructAuto.proxyHostName		NULL terminated string representing the SIP Proxy Hostname

sStructAuto.bUseProxyForRegistration	This refers that SIP uses Proxy For Registration or Direct mode	
	direct=0	Direct mode
	proxy=1	Through Proxy
sStructAuto.transport	SIP_TRANSPORT_TYPE_AUTO=0	SIP_TRANSPORT_TYPE_AUTO
	SIP_TRANSPORT_TYPE_UDP_ONLY=1	SIP_TRANSPORT_TYPE_UDP_ONLY
	SIP_TRANSPORT_TYPE_TCP_ONLY=2	SIP_TRANSPORT_TYPE_TCP_ONLY
	SIP_TRANSPORT_TYPE_TLS_ONLY=3	SIP_TRANSPORT_TYPE_TLS_ONLY
sStructOCSManual	This struct needs to be filled up only in case the eSipServerType is having value of SIP_SERVER_OCS_MANUAL	
sStructOCSManual.bIsTransportTCP	This refers that TCP enabled or disabled.	
	tls=0	TCP is NOT enabled
	tcp=1	TCP is enabled
sStructOCSManual.internalServer	NULL terminated string representing the internal server	
sStructOCSManual.externalServer	NULL terminated string representing the external server	

eContextType *unsigned int* — This decides whether this refers to the primary or the secondary SIP configuration

SIP_CONTEXT_PRIMARY=0	SIP_CONTEXT_PRIMARY
SIP_CONTEXT_SECONDARY=1	SIP_CONTEXT_SECONDARY

Outputs

_rv *integer* — commReturnVal

COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
COMMRPC_FAILURE=-601	COMMRPC_FAILURE
COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL
COMMRPC_FAILURE_NOTALLOWED_OCS=-607	COMMRPC_FAILURE_NOTALLOWED_OCS

SYNC getSipDetails(SipDetails *pSipDetails,SipContextType eContextType)

This is used to get the SIP details for SIP configuration

privilege level ADMIN

Parameters

Inputs

eContextType *unsigned int* — This decides whether this refers to the primary or the secondary SIP configuration

SIP_CONTEXT_PRIMARY=0	SIP_CONTEXT_PRIMARY
SIP_CONTEXT_SECONDARY=1	SIP_CONTEXT_SECONDARY

Outputs

_rv *integer* — Return status (0 = success)

pSipDetails *structure* — This structure contains the actual SIP details

bIsSipEnabled	disabled=0	SIP is NOT enabled
	enabled=1	SIP is enabled
userName	NULL terminated string representing the UserName	
authName	NULL terminated string representing the authName	
authPassword	NULL terminated string representing the authPassword	
eSipServerType	SIP_SERVER_AUTO=0	SIP_SERVER_AUTO
	SIP_SERVER_OCS=1	SIP_SERVER_OCS
	SIP_SERVER_OCS_MANUAL=2	SIP_SERVER_OCS_MANUAL
sStructAuto	This struct needs to be filled up only in case the eSipServerType is having value of SIP_SERVER_AUTO	
sStructAuto.bIsRegistrarEnabled	This refers that SIP Registrar is disabled or enabled	
	disabled=0	Registrar is disabled
	enabled=1	Registrar is enabled
sStructAuto.registrarHostName	NULL terminated string representing the SIP Registrar Hostname	
sStructAuto.bIsProxyEnabled	This refers that SIP Proxy is disabled or enabled	
	disabled=0	Proxy is disabled

	enabled=1 Proxy is enabled
sStructAuto.proxyHostName	NULL terminated string representing the SIP Proxy Hostname
sStructAuto.bUseProxyForRegistration	This refers that SIP uses Proxy For Registration or Direct mode direct=0 Direct mode proxy=1 Through Proxy
sStructAuto.transport	SIP_TRANSPORT_TYPE_AUTO=0 SIP_TRANSPORT_TYPE_AUTO SIP_TRANSPORT_TYPE_UDP_ONLY=1 SIP_TRANSPORT_TYPE_UDP_ONLY SIP_TRANSPORT_TYPE_TCP_ONLY=2 SIP_TRANSPORT_TYPE_TCP_ONLY SIP_TRANSPORT_TYPE_TLS_ONLY=3 SIP_TRANSPORT_TYPE_TLS_ONLY
sStructOCSManual	This struct needs to be filled up only in case the eSipServerType is having value of SIP_SERVER_OCS_MANUAL
sStructOCSManual.bIsTransportTCP	This refers that TCP enabled or disabled. tls=0 TCP is NOT enabled tcp=1 TCP is enabled
sStructOCSManual.internalServer	NULL terminated string representing the internal server
sStructOCSManual.externalServer	NULL terminated string representing the external server

EVENT sipDetailsUpdated(int bIsSipEnabled, SipContextType eContextType)

This is the event that is sent by comm to indicate that SIP configuration details have been modified

Parameters

bIsSipEnabled	*integer*	disabled=0 SIP is NOT enabled enabled=1 SIP is enabled
eContextType	*unsigned int*	This decides whether this refers to the primary or the secondary SIP configuration SIP_CONTEXT_PRIMARY=0 SIP_CONTEXT_PRIMARY SIP_CONTEXT_SECONDARY=1 SIP_CONTEXT_SECONDARY

SYNC sipRegister(SipContextType eContextType)

This is the method that can be used to force a registration.

privilege level ADMIN

Parameters

Inputs

eContextType	*unsigned int*	This decides whether this refers to the primary or the secondary SIP configuration SIP_CONTEXT_PRIMARY=0 SIP_CONTEXT_PRIMARY SIP_CONTEXT_SECONDARY=1 SIP_CONTEXT_SECONDARY

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getSipRegisterStatus(RegistrationStatus *peStatus,SipContextType eContextType)

This is invoked to obtain the current SIP registration status

privilege level ADMIN

Parameters

Inputs

eContextType	*unsigned int*	This decides whether this refers to the primary or the secondary SIP configuration SIP_CONTEXT_PRIMARY=0 SIP_CONTEXT_PRIMARY SIP_CONTEXT_SECONDARY=1 SIP_CONTEXT_SECONDARY

Outputs

_rv	*integer*	Return status (0 = success)
peStatus	*unsigned int*	This contains the actual registration status

REG_STATUS_REGISTERED=0	REG_STATUS_REGISTERED
REG_STATUS_REGISTERING=1	REG_STATUS_REGISTERING
REG_STATUS_UNREGISTERED=2	REG_STATUS_UNREGISTERED
REG_STATUS_UNREACHABLE=3	REG_STATUS_UNREACHABLE
REG_STATUS_FAILED=4	REG_STATUS_FAILED

EVENT sipRegisterStatusUpdated(RegistrationStatus eStatus,SipContextType eContextType)

This is thrown by comm to indicate that the registration status has been updated.

Parameters

eStatus *unsigned int* This contains the actual registration status.

REG_STATUS_REGISTERED=0	REG_STATUS_REGISTERED
REG_STATUS_REGISTERING=1	REG_STATUS_REGISTERING
REG_STATUS_UNREGISTERED=2	REG_STATUS_UNREGISTERED
REG_STATUS_UNREACHABLE=3	REG_STATUS_UNREACHABLE
REG_STATUS_FAILED=4	REG_STATUS_FAILED

eContextType *unsigned int* This decides whether this refers to the primary or the secondary SIP configuration

SIP_CONTEXT_PRIMARY=0	SIP_CONTEXT_PRIMARY
SIP_CONTEXT_SECONDARY=1	SIP_CONTEXT_SECONDARY

SYNC setLSTransitDetails(const LSTransitDetails *pLSTransitDetails)

This is used to set the LSTransit details

privilege level ADMIN

Parameters

Inputs

pLSTransitDetails *structure* This struct contains the LSTransit details

bIsLSTransitEnabled		
	disabled=0	Transit is NOT enabled
	enabled=1	Transit is enabled
transitHostName	NULL terminated string indicating the LSTransitHostName	
transitUserName	NULL terminated string indicating the LSTransitUserName	
transitPassword	NULL terminated string indicating the LSTransitPassword	
bIsLSTransitEnabledForSip	This field decides if we need to use LSTransit for SIP	
	disabled=0	Transit is NOT enabled for SIP
	enabled=1	Transit is enabled for SIP
sipUserName	NULL terminated string that represents the SIP user name to be used to REGISTER with LSTransit. This needs to be available only if bIsLSTransitEnabledForSip is true	
bIsLSTransitEnabledForH323	This field decides if we need to use LSTransit for H323.	
	disabled=0	Transit is NOT enabled for H.323
	enabled=1	Transit is enabled for H.323
strH323Extn	NULL terminated string that represents the H323 Extension to be used when H323 is used with LSTransit. This needs to be available only if bIsLSTransitEnabledForH323 is true	

Outputs

_rv *integer* commReturnVal

COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
COMMRPC_FAILURE=-601	COMMRPC_FAILURE
COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
COMMRPC_FAILURE_NOTALLOWED_H323_IS_ENABLED=-604	COMMRPC_FAILURE_NOTALLOWED_H323_IS_ENABLED
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getLSTransitDetails(LSTransitDetails *pLSTransitDetails)

This is used to obtain the currently set LSTransit details

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pLSTransitDetails	*structure*	This struct contains the LSTransit details

bIsLSTransitEnabled	disabled=0 Transit is NOT enabled enabled=1 Transit is enabled
transitHostName	NULL terminated string indicating the LSTransitHostName
transitUserName	NULL terminated string indicating the LSTransitUserName
transitPassword	NULL terminated string indicating the LSTransitPassword
bIsLSTransitEnabledForSip	This field decides if we need to use LSTransit for SIP disabled=0 Transit is NOT enabled for SIP enabled=1 Transit is enabled for SIP
sipUserName	NULL terminated string that represents the SIP user name to be used to REGISTER with LSTransit. This needs to be available only if bIsLSTransitEnabledForSip is true
bIsLSTransitEnabledForH323	This field decides if we need to use LSTransit for H323. disabled=0 Transit is NOT enabled for H.323 enabled=1 Transit is enabled for H.323
strH323Extn	NULL terminated string that represents the H323 Extension to be used when H323 is used with LSTransit. This needs to be available only if bIsLSTransitEnabledForH323 is true

`EVENT lsTransitDetailsUpdated(int bIsLSTransitEnabled, int bIsSipTunneled, int bIsH323Tunneled)`

This is an event thrown to indicate that the LS Transit details have been updated

Parameters

bIsLSTransitEnabled	*integer*	This refers the status of LS Transit, 0 - Transit is NOT enabled and 1 - Transit is enabled disabled=0 Transit is NOT enabled enabled=1 Transit is enabled
bIsSipTunneled	*integer*	This refers the status of LS SIP Tunnel, 0 - SIP Tunnel is NOT enabled and 1 - SIP Tunnel is enabled disabled=0 Transit is NOT enabled for SIP enabled=1 Transit is enabled for SIP
bIsH323Tunneled	*integer*	This refers the status of LS H323 Tunnel, 0 - H323 Tunnel is NOT enabled and 1 - H323 Tunnel is enabled disabled=0 Transit is NOT enabled for H.323 enabled=1 Transit is enabled for H.323

`SYNC getLSTransitStatus(LSTransitStatus *lsTransitStatus, RegistrationStatus *sipTransitStatus, RegistrationStatus *h323TransitStatus)`

This is used to obtain the current LSTransit connectivity status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
lsTransitStatus	*unsigned int*	This conveys the current LSTransit connectivity status disabled=0 is Disabled connected=1 is Connected failed=2 is Failed
sipTransitStatus	*unsigned int*	This conveys the current SIP Transit Registration Status REG_STATUS_REGISTERED=0 REG_STATUS_REGISTERED REG_STATUS_REGISTERING=1 REG_STATUS_REGISTERING REG_STATUS_UNREGISTERED=2 REG_STATUS_UNREGISTERED REG_STATUS_UNREACHABLE=3 REG_STATUS_UNREACHABLE

REG_STATUS_FAILED=4 REG_STATUS_FAILED

h323TransitStatus *unsigned int* This conveys the current H323 Transit Registration Status

REG_STATUS_REGISTERED=0	REG_STATUS_REGISTERED
REG_STATUS_REGISTERING=1	REG_STATUS_REGISTERING
REG_STATUS_UNREGISTERED=2	REG_STATUS_UNREGISTERED
REG_STATUS_UNREACHABLE=3	REG_STATUS_UNREACHABLE
REG_STATUS_FAILED=4	REG_STATUS_FAILED

EVENT lsTransitStatusUpdated(LSTransitStatus lsTransitStatus, RegistrationStatus sipTransitStatus, RegistrationStatus h323TransitStatus)

This event is thrown by comm whenever the LSTransit connectivity status changes.

Parameters

lsTransitStatus *unsigned int* This conveys the current LSTransit connectivity status

disabled=0	is Disabled
connected=1	is Connected
failed=2	is Failed

sipTransitStatus *unsigned int* This conveys the current SIP Transit Registration Status

REG_STATUS_REGISTERED=0	REG_STATUS_REGISTERED
REG_STATUS_REGISTERING=1	REG_STATUS_REGISTERING
REG_STATUS_UNREGISTERED=2	REG_STATUS_UNREGISTERED
REG_STATUS_UNREACHABLE=3	REG_STATUS_UNREACHABLE
REG_STATUS_FAILED=4	REG_STATUS_FAILED

h323TransitStatus *unsigned int* This conveys the current H323 Transit Registration Status

REG_STATUS_REGISTERED=0	REG_STATUS_REGISTERED
REG_STATUS_REGISTERING=1	REG_STATUS_REGISTERING
REG_STATUS_UNREGISTERED=2	REG_STATUS_UNREGISTERED
REG_STATUS_UNREACHABLE=3	REG_STATUS_UNREACHABLE
REG_STATUS_FAILED=4	REG_STATUS_FAILED

EVENT ConnectionsEnableUpdated(int isConnectionsEnabled)

Notification of Connections support status

Parameters

isConnectionsEnabled *integer* This event returns Connections is enabled or not. 1=Enabled,0=Disabled

EVENT registerToConnectionsTriggered(ConstStringZ connectionsUserId, ConstStringZ registrationKey)

This event thrown when perform manual connections configuration. Event is mainly towards the lscconfapp for triggering manual configuration

Parameters

connectionsUserId *string* Denotes the user id to be used during manual configuration

registrationKey *string* Denotes the one time valid key that should be used for registering with the service for the first time (in manual mode)

EVENT connectionsStatusUpdated(ConnectionsStatus status)

at any point if the connections status changes this event is thrown.

Parameters

status *unsigned int* This event indicate the connections status and Its an enum

0	CONN_STATUS_NONE
1	CONN_STATUS_CONNECTED
2	CONN_STATUS_DISCONNECTED
3	CONN_STATUS_AUTHENTICATION_UNNECESSARY
4	CONN_STATUS_INVALID_CONNECTIONS_ID

5 CONN_STATUS_INVALID_REGISTRATION_KEY
6 CONN_STATUS_SERVICE_UNAVAILABLE
7 CONN_STATUS_SERVER_UNREACHABLE
8 CONN_STATUS_SIP_DISABLED
9 CONN_STATUS_SUBSCRIPTION_EXPIRED
10 CONN_STATUS_UNREGISTERED_SYSTEM
11 CONN_STATUS_CONNECTING
12 CONN_STATUS_DISCONNECTING

EVENT refreshContactList()

on receiving a MESSAGE request with the json body indicating refresh_xcap comm would throw this event to lscconfapp to download the contact list

EVENT refreshDirectoryEntries()

on receiving a MESSAGE request with the json body indicating refresh_smb_directory comm would throw this event to lscconfapp to download the corpoeate directory entries

EVENT incomingCall(unsigned int commCallId, ConstStringZ callingPartyDisplayName, ConstStringZ callingPartyAddress, Protocol protocol)

This is generated by comm when it receives an incoming call. UI needs to accept/reject the call.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call. UI needs to quote this to refer to this call here onwards.
callingPartyDisplayName	*string*	This is the name of the callingParty and can be displayed by UI as x is calling
callingPartyAddress	*string*	This is the calling party address . Primarily of use in populating the redial list.
protocol	*unsigned int*	This is the protocol over which the call was received 0 PROTOCOL_AUTO 1 PROTOCOL_H323 2 PROTOCOL_SIP 3 PROTOCOL_RTSP 4 PROTOCOL_SIP_LSC

EVENT callRinging(unsigned int commCallId, int playRingBackTone)

Thrown to indicate, remote party's phone is ringing. This can come more than once for the ringback control.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
playRingBackTone	*integer*	This informs UI whether or not to play the ringing tone. 0 means do not play, 1 means play

EVENT callConnected(unsigned int commCallId, unsigned int commConfId, int bVirtualCall, Protocol protocol, ConstStringZ remotePartyDisplayName, ConstStringZ remotePartyAddress, RemoteVendorInfoUI* pVendorInfo,int bIsCallSecure, int bIsPresentationSupported)

This is thrown by comm to indicate that signallingwise call is connected.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
commConfId	*unsigned int*	This indicated the conference to which this call belongs.
bVirtualCall	*integer*	If bVirtualCall==1, it means the call is remotely hosted, no resources locally are being consumed (like audio/video decode/encode). Scenarios here are lsconnections and call escalation to He. This piece of information is useful since UI might need to do special handling for these calls. One thing can be that statistics for this kind of call might not be available. Also it might be required in some cases that the end user not be able to disconnect this call.
protocol	*unsigned int*	The protocol for the call. SIP/323/LSC etc. 0 PROTOCOL_AUTO

1 PROTOCOL_H323
2 PROTOCOL_SIP
3 PROTOCOL_RTSP
4 PROTOCOL_SIP_LSC

remotePartyDisplayName	*string*	This is the displayName of the remote party.
remotePartyAddress	*string*	This is the address of the remote party
pVendorInfo	*structure*	gives remote vendor related information like vendorname,version number etc.
bIsCallSecure	*integer*	Security status. Unsecure - 0, Secure - 1
bIsPresentationSupported	*integer*	Presentation status. Not-Supported - 0, Supported - 1

EVENT callTransferred(unsigned int oldCommCallId, unsigned int newCommCallId, ConstStringZ remotePartyDisplayName)

This message is thrown if the oldCommCallId is replaced by newCommCallId.

Parameters

oldCommCallId	*unsigned int*	This is how comm refers to the existing call
newCommCallId	*unsigned int*	This is how comm refers to the new call
remotePartyDisplayName	*string*	This is the displayName of the new remote party

EVENT changeCallCharacteristics(unsigned int commCallId, int bUpdateRedialList, Protocol protocol, ConstStringZ remotePartyDisplayName, ConstStringZ remotePartyAddress, RemoteVendorInfoUI* pVendorInfo,int bIsCallSecure, int bIsPresentationSupported)

This is thrown by comm to change some properties related to the i/c or o/g call. This allows comm to control the redial list entry and also to change the protocol type in some cases.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
bUpdateRedialList	*integer*	This is how comm requests to not add any entry to redial list for this call. 0 means do not update. By default 1 do update.
protocol	*unsigned int*	This is the new protocol. Sometimes a SIP call becomes a LSC call. 0 PROTOCOL_AUTO 1 PROTOCOL_H323 2 PROTOCOL_SIP 3 PROTOCOL_RTSP 4 PROTOCOL_SIP_LSC
remotePartyDisplayName	*string*	This is the new display name that comm desires for the remote party. If empty then retain whatever was present.
remotePartyAddress	*string*	This is the new address that comm desires for the remote party. If empty then retain whatever was present
pVendorInfo	*structure*	gives remote vendor related information like vendorname,version number etc.
bIsCallSecure	*integer*	Security status. Unsecure - 0, Secure - 1
bIsPresentationSupported	*integer*	Presentation status. Not-Supported - 0, Supported - 1

EVENT mediaChannelDisconnected(unsigned int commCallId, MediaDirection channelDirection, MediaType mediaType)

Event generated by comm to indicate that media on a particular call in a direction is now disconnected.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
channelDirection	*unsigned int*	This is used to say whether the channel is incoming or outgoing or the encode/decode direction 0 TRANSMIT 1 RECEIVE 2 BIDIRECTIONAL
mediaType	*unsigned int*	This is the type of media, audio/video/fecc/ancillary etc 0 MEDIA_AUDIO = 1 1 MEDIA_VIDEO 2 MEDIA_DATA 3 MEDIA_ANCILLARY_VIDEO 4 MEDIA_STREAMING 5 MEDIA_TYPE_MAX

EVENT mediaChannelConnected(unsigned int commCallId, MediaDirection channelDirection, MediaType mediaType, int bandwidth, int deviceId)

Event generated by comm to indicate that media on a particular call in a direction is properly connected.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
channelDirection	*unsigned int*	This is used to say whether the channel is incoming or outgoing or the encode/decode direction 0 TRANSMIT 1 RECEIVE 2 BIDIRECTIONAL
mediaType	*unsigned int*	This is the type of media, audio/video/fecc/ancillary etc. 0 MEDIA_AUDIO = 1 1 MEDIA_VIDEO 2 MEDIA_DATA 3 MEDIA_ANCILLARY_VIDEO 4 MEDIA_STREAMING 5 MEDIA_TYPE_MAX
bandwidth	*integer*	This is the bandwidth for the channel.
deviceId	*integer*	Apllies for audio/video related channels only. Refers to the channelId that comm gets from audio/video

EVENT mediaChannelChanged(unsigned int commCallId, MediaDirection channelDirection, MediaType mediaType, int deviceId)

Notifies UI about deviceId change on mediaChannel, for instance when we switch decoders on the fly in case of SIP.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
channelDirection	*unsigned int*	This is used to say whether the channel is incoming or outgoing or the encode/decode direction 0 TRANSMIT 1 RECEIVE 2 BIDIRECTIONAL
mediaType	*unsigned int*	This is the type of media, audio/video/fecc/ancillary etc. 0 MEDIA_AUDIO = 1 1 MEDIA_VIDEO 2 MEDIA_DATA 3 MEDIA_ANCILLARY_VIDEO 4 MEDIA_STREAMING 5 MEDIA_TYPE_MAX
deviceId	*integer*	media channel device Identifier

EVENT terminatingCall(unsigned int commCallId, CallDisconnectReason reasonCode)

This is generated by comm to indicate that a call is in the process of disconnection.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
reasonCode	*unsigned int*	This indicates the reason for the disconnection 0 CALL_DISC_REASON_NONE 1 CALL_DISC_REASON_NORMAL_DISCONNECT 2 CALL_DISC_REASON_USER_BUSY 3 CALL_DISC_REASON_UNREACHABLE_DESTINATION 4 CALL_DISC_REASON_DESTINATION_REJECTION 5 CALL_DISC_REASON_FACILITY_CALL_DEFLECTION 6 CALL_DISC_REASON_INCONF 7 CALL_DISC_REASON_NO_BANDWIDTH 8 CALL_DISC_REASON_SECURITY_DENIED

9	CALL_DISC_REASON_TCS_REJECTED_BY_PEER
10	CALL_DISC_REASON_LOCAL_FAILURE
11	CALL_DISC_REASON_UNREACHABLE_GK
12	CALL_DISC_REASON_GK_RESOUCES_UNAVAILABLE
13	CALL_DISC_REASON_GW_RESOUCES_UNAVAILABLE
14	CALL_DISC_REASON_INVALID_ADDRESS
15	CALL_DISC_REASON_CALLEDPARTY_NOTREGISTERED
16	CALL_DISC_REASON_CALLER_NOTREGISTERED
17	CALL_DISC_REASON_USER_MAX_CALLS_EXCEEDED
18	CALL_DISC_REASON_AUDIO_RESOURCE_UNAVAILABLE
19	CALL_DISC_REASON_MEDIA_SERVER_UNAVAILABLE
20	CALL_DISC_REASON_UI_DIED
21	CALL_DISC_REASON_UNDEFINED

`EVENT remoteCallDisconnected(unsigned int commCallId, CallDisconnectReason reasonCode)`

This is generated by comm to indicate that a call has been disconnected.

Parameters

`commCallId` *unsigned int* This is how comm refers to the call.

`reasonCode` *unsigned int* This indicates the reason for the disconnection

0	CALL_DISC_REASON_NONE
1	CALL_DISC_REASON_NORMAL_DISCONNECT
2	CALL_DISC_REASON_USER_BUSY
3	CALL_DISC_REASON_UNREACHABLE_DESTINATION
4	CALL_DISC_REASON_DESTINATION_REJECTION
5	CALL_DISC_REASON_FACILITY_CALL_DEFLECTION
6	CALL_DISC_REASON_INCONF
7	CALL_DISC_REASON_NO_BANDWIDTH
8	CALL_DISC_REASON_SECURITY_DENIED
9	CALL_DISC_REASON_TCS_REJECTED_BY_PEER
10	CALL_DISC_REASON_LOCAL_FAILURE
11	CALL_DISC_REASON_UNREACHABLE_GK
12	CALL_DISC_REASON_GK_RESOUCES_UNAVAILABLE
13	CALL_DISC_REASON_GW_RESOUCES_UNAVAILABLE
14	CALL_DISC_REASON_INVALID_ADDRESS
15	CALL_DISC_REASON_CALLEDPARTY_NOTREGISTERED
16	CALL_DISC_REASON_CALLER_NOTREGISTERED
17	CALL_DISC_REASON_USER_MAX_CALLS_EXCEEDED
18	CALL_DISC_REASON_AUDIO_RESOURCE_UNAVAILABLE
19	CALL_DISC_REASON_MEDIA_SERVER_UNAVAILABLE
20	CALL_DISC_REASON_UI_DIED
21	CALL_DISC_REASON_UNDEFINED

`EVENT remoteCallId(unsigned int commCallId, unsigned int remoteCallId)`

This is generated by comm to indicate helium/bridge callid of commCallId.

Parameters

`commCallId` *unsigned int* This is how comm refers to the call.

`remoteCallId` *unsigned int* callid in helium/bridge for corrsosponding call leg for the tight integrtion feature.

`EVENT dtmfRecvd(unsigned int commCallId, char digit)`

This is the event that is generated by comm to indicate receipt of a digit over the DTMF.

Parameters

`commCallId`	*unsigned int*	This is how comm refers to the call.
`digit`	*unsigned int*	The DTMF digit that was received

`EVENT remotePresentationStarted(unsigned int commConfId,unsigned int commCallId)`

This is the event generated by comm to indicate that remote side has started presentation.

Parameters

`commConfId`	*unsigned int*	Conference in which this call is present. On 300s this will always be the default conference.
`commCallId`	*unsigned int*	This is how comm refers to the call

`EVENT remotePresentationStopped(unsigned int commConfId,unsigned int commCallId)`

This is the event generated by comm to indicate that remote side has stopped presentation.

Parameters

`commConfId`	*unsigned int*	Conference in which this call is present. On 300s this will always be the default conference.
`commCallId`	*unsigned int*	This is how comm refers to the call

`EVENT presentationStateChanged(unsigned int commConfId, unsigned int commCallId, PresentationStates state)`

event thrown by comm to indicate that the presentation state has changed. This is a common event thrown for both local and remote presentation streams

Parameters

`commConfId`	*unsigned int*	Conference in which this call is present. On 300s this will always be the default conference.
`commCallId`	*unsigned int*	This is how comm refers to the call
`state`	*unsigned int*	indicates the state of the presentation stream long with the direction 0 STOPPED 1 REMOTE_STARTED 2 REMOTE_STARTING 3 REMOTE_STOPPING 4 LOCAL_STARTED 5 LOCAL_STARTING 6 LOCAL_STOPPING

`EVENT farAudioMuteStatusChanged(unsigned int commCallId, AudioMuteStatus farMuteStatus)`

This is an event thrown by comm to indicate that the remote side has muted/unmuted the audio. This is sent on a per call basis.

Parameters

`commCallId`	*unsigned int*	This is how comm refers to the call
`farMuteStatus`	*unsigned int*	This informs whether the remote end's audio is muted/unmuted. 0-muted & 1-unmuted

`EVENT feccCapabilityRecvd(unsigned int commCallId,FeccCapsMsg *psFeccCapsMsg)`

This is the event that comm throws when it is aware of the remote end's FECC capability

Parameters

`commCallId`	*unsigned int*	This is how comm refers to the call.
`psFeccCapsMsg`	*structure*	This struct has the relevant fields about remote end's FECC capability uchNumPresets — Number of presets supported by the endpoint. This structure member shall be used when the camera presets are supported in comm uchNumVidSrcs — total number of video sources that are supported by the endpoint. VideoSrcInfo.FeccVideoSource — pointer to an array of video source information. The length of this array = uchNumVidSrcs 255 FECC_INVALID_VIDEO_SRC = 255 0 FECC_DEFAULT_CAMERA = 0

1 FECC_MAIN_CAMERA = 1
2 FECC_AUX_CAMERA = 2
3 FECC_DOC_CAMERA = 3
4 FECC_AUX_DOC_CAMERA = 4
5 FECC_VID_PLAY_BACK_SRC = 5
6 FECC_EXTENDED_VID_SRC_1 = 6
7 FECC_EXTENDED_VID_SRC_2 = 7
8 FECC_EXTENDED_VID_SRC_3 = 8
9 FECC_EXTENDED_VID_SRC_4 = 9
10 FECC_EXTENDED_VID_SRC_5 = 10
11 FECC_EXTENDED_VID_SRC_6 = 11
12 FECC_EXTENDED_VID_SRC_7 = 12
13 FECC_EXTENDED_VID_SRC_8 = 13
14 FECC_EXTENDED_VID_SRC_9 = 14
15 FECC_EXTENDED_VID_SRC_10 = 15
16 FECC_MAX_NUM_VID_SRCS

VideoSrcInfo.uchCamPTZFCaps camera pan, tilt, zoom, focus capability support
0 CAM_CTRL_CMD_NONE =0
1 PAN_LEFT
2 PAN_RIGHT
3 TILT_UP
4 TILT_DOWN
5 ZOOM_IN
6 ZOOM_OUT
7 FOCUS_IN
8 FOCUS_OUT
9 CONTINUE

EVENT feccPresetCmdRecvd(unsigned int commCallId, unsigned int uiPresetPosition)

This is the event that comm throws when it receives a request from remote end to switch to a particular preset.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
uiPresetPosition	*unsigned int*	This is the preset position to which the remote end wants our camera source to switch to

EVENT feccMsgRecvd(unsigned int commCallId,CameraFarEndMsg *psFarEndMsg)

This is the event that comm throws to indicate receipt of a FECC message from remote side.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
psFarEndMsg	*structure*	This struct is the particular message that conveys what the remote side wants
	eCameraType	Camera Type NONE = 0, FAR_END=1, NEAR_END=2
	eAction	Camera action CAM_CTRL_ACTION_NONE=0, MOVE=1, STOP=2, CAM_CTRL_ACTION_CHANGE_SRC=3
	eCmd	camera pan, tilt, zoom, focus capability support
		0 CAM_CTRL_CMD_NONE =0
		1 PAN_LEFT
		2 PAN_RIGHT
		3 TILT_UP
		4 TILT_DOWN
		5 ZOOM_IN
		6 ZOOM_OUT
		7 FOCUS_IN
		8 FOCUS_OUT

uchCamSourceNo	camera source number
uchTimeout	Time out in seconds

EVENT feccSetPresetCmdRecvd(unsigned int commCallId, unsigned int uiPresetPosition)

This is the Event comm throws when remote end want to set preset locally

Parameters

commCallId	*unsigned int*	This is how comm refers to the call
uiPresetPosition	*unsigned int*	This is the new position that remote end want to set to

EVENT statsEvent(unsigned int commCallId,UiCallStats *psuiCallStat)

This is the call stats event thrown with all the call details

Parameters

commCallId	*unsigned int*	This is how comm refers to the call
psuiCallStat	*structure*	This struct gives the all call details

SYNC setDualVideoFeature(unsigned int bEnabled)

This is the API using which comm is told to enable/disable dual video feature

privilege level ADMIN

Parameters

Inputs

bEnabled	*unsigned int*	If bEnabled=1, enable dual video feature, if bEnabled=0, disable dual video feature

Outputs

_rv	*integer*	commReturnVal

COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
COMMRPC_FAILURE=-601	COMMRPC_FAILURE
COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getDualVideoFeature(unsigned int *pEnabled)

This is the API using which anyone can determine if dual video feature is currently enabled

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pEnabled	*unsigned int*	This is a return param

disabled=0	dual video is disabled
enabled=1	dual video is enabled

EVENT dualVideoFeatureUpdated(unsigned int bEnabled)

This is an event that is thrown to indicate that there is change in dualVideo feature

Parameters

bEnabled	*unsigned int*	This is a return param

disabled=0	dual video is disabled
enabled=1	dual video is enabled

SYNC setFarControlPropertyOverLocalCamera(const FarControlOnLocalCamera *psProperty)

This is the API using which comm is told to apply the rules related to far end control of local camera

privilege level ADMIN

Parameters

Inputs

psProperty	*structure*	This contains the latest applicable properties
	bAllowFarToControlLocalCamera	This decides whether the far end can control the local camera, PTZ etc.
	disabled=0	far end can control the local camera is disabled
	enabled=1	far end can control the local camera is enabled
	bAllowFarToChangeLocalPresets	This decides whether the far end can set the local camera presets.
	disabled=0	far end can set the local camera presets is disabled
	enabled=1	far end can set the local camera presets is enabled
	bAllowFarToSwitchToLocalPresets	This decides whether the far end can modify/update the local camera presets.
	disabled=0	far end can modify/update the local camera presets is disabled
	enabled=1	far end can modify/update the local camera presets is enabled

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getFarControlPropertyOverLocalCamera(FarControlOnLocalCamera *pProperty)

This is the API using which anyone can determine the rules related to far end control of local camera

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pProperty	*structure*	This contains the latest applicable properties
	bAllowFarToControlLocalCamera	This decides whether the far end can control the local camera, PTZ etc.
	disabled=0	far end can control the local camera is disabled
	enabled=1	far end can control the local camera is enabled
	bAllowFarToChangeLocalPresets	This decides whether the far end can set the local camera presets.
	disabled=0	far end can set the local camera presets is disabled
	enabled=1	far end can set the local camera presets is enabled
	bAllowFarToSwitchToLocalPresets	This decides whether the far end can modify/update the local camera presets.
	disabled=0	far end can modify/update the local camera presets is disabled
	enabled=1	far end can modify/update the local camera presets is enabled

EVENT farControlPropertyOnLocalCameraUpdated()

This is thrown to indicate that local rules related to far end control over local camera have been modified

SYNC setVideoMtuCfg(unsigned int uiCfgVal)

This is used to inform comm to assume a particular MTU value for the video packets.

privilege level ADMIN

Parameters

Inputs

uiCfgVal	*unsigned int*	This is the actual MTU value that comm needs to assume

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getVideoMtuCfg(unsigned int *pUiCfgVal)

This can be used to query comm about the latest MTU value it applies for o/b video packets.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pUiCfgVal	*unsigned int*	This is the actual MTU value

EVENT videoMtuCfgUpdated(unsigned int uiCfgVal)

This is an event thrown by comm to indicate that the video MTU value has been modified.

Parameters

uiCfgVal	*unsigned int*	This is the actual MTU value

SYNC setQos(QosMode eQosMode, short audioPriority, short videoPriority, short dataPriority, short tos)

This is used to inform comm about the QoS params that it needs to apply.

privilege level ADMIN

Parameters

Inputs

eQosMode	*unsigned int*	This can be a enum value for OFF or DiffServ or IP_PRECEDENCE(IntServ) IP_PRECEDENCE=0 IP_PRECEDENCE DIFFSERV=1 DIFFSERV QOSMODE_OFF=2 QOSMODE_OFF
audioPriority	*integer*	This specifies the Audio priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
videoPriority	*integer*	This specifies the video priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
dataPriority	*integer*	This specifies the data priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
tos	*integer*	This specifies the ToS if we have selected IntServ/IP_PRECEDENCE for the eQoSmode TOS_NONE=0 TOS_NONE TOS_MAXIMUM_THROUGHPUT=1 TOS_MAXIMUM_THROUGHPUT TOS_MINIMUM_DELAY=2 TOS_MINIMUM_DELAY TOS_MAXIMUM_THERSHOLD=3 TOS_MAXIMUM_THERSHOLD TOS_MINIMUM_COST=4 TOS_MINIMUM_COST

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getQos(QosMode *eQosMode, short *audioPriority, short *videoPriority, short *dataPriority, short *tos)

This is used to query comm about the QoS params that it currently applies.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
eQosMode	*unsigned int*	This can be a enum value for OFF or DiffServ or IP_PRECEDENCE(IntServ) IP_PRECEDENCE=0 IP_PRECEDENCE DIFFSERV=1 DIFFSERV QOSMODE_OFF=2 QOSMODE_OFF
audioPriority	*integer*	This specifies the Audio priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
videoPriority	*integer*	This specifies the video priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
dataPriority	*integer*	This specifies the data priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63

tos	*integer*	This specifies the ToS if we have selected IntServ/IP_PRECEDENCE for the eQoSmode TOS_NONE=0 TOS_NONE TOS_MAXIMUM_THROUGHPUT=1 TOS_MAXIMUM_THROUGHPUT TOS_MINIMUM_DELAY=2 TOS_MINIMUM_DELAY TOS_MAXIMUM_THERSHOLD=3 TOS_MAXIMUM_THERSHOLD TOS_MINIMUM_COST=4 TOS_MINIMUM_COST

SYNC getQosExt(QosMode eQosMode, short *audioPriority, short *videoPriority, short *dataPriority, short *tos)

This is used to query comm about the particular QosMode values. Like Audio/Video/Data priority values for the IP_PRECEDENCE only. Audio / Video / Data priority values for the DIFFSERV only. Tos applicable only for IP_PRECEDENCE

privilege level ADMIN

Parameters

Inputs

eQosMode	*unsigned int*	This can be a enum value for OFF or DiffServ or IP_PRECEDENCE(IntServ) IP_PRECEDENCE=0 IP_PRECEDENCE DIFFSERV=1 DIFFSERV QOSMODE_OFF=2 QOSMODE_OFF

Outputs

_rv	*integer*	Return status (0 = success)
audioPriority	*integer*	This specifies the Audio priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
videoPriority	*integer*	This specifies the video priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
dataPriority	*integer*	This specifies the data priority. If eQoSmode=IP_PRECEDENCE it should be set only 0-7, If eQoSmode=DIFFSERV it should be set only 0-63
tos	*integer*	This specifies the ToS if we have selected IntServ/IP_PRECEDENCE for the eQoSmode TOS_NONE=0 TOS_NONE TOS_MAXIMUM_THROUGHPUT=1 TOS_MAXIMUM_THROUGHPUT TOS_MINIMUM_DELAY=2 TOS_MINIMUM_DELAY TOS_MAXIMUM_THERSHOLD=3 TOS_MAXIMUM_THERSHOLD TOS_MINIMUM_COST=4 TOS_MINIMUM_COST

EVENT qosUpdated()

This event is thrown when the qos is updated

SYNC setSipSecurityProperty(const SecurityType value)

This is the method using which comm can be influenced to apply SIP security settings on the calls

privilege level ADMIN

Parameters

Inputs

value	*unsigned int*	This is the struct that has the relevant SIP security level fields comm is interested in SECURITY_AUTO=0 for SECURITY_AUTO SECURITY_OFF=1 for SECURITY_OFF SECURITY_STRICT=2 for SECURITY_STRICT

Outputs

_rv	*integer*	commReturnVal COMMRPC_SUCCESS=0 COMMRPC_SUCCESS COMMRPC_FAILURE=-601 COMMRPC_FAILURE COMMRPC_FAILURE_INVALIDPARAM=-602 COMMRPC_FAILURE_INVALIDPARAM COMMRPC_FAILURE_NOTALLOWED_INCALL=-606 COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getSipSecurityProperty(SecurityType *value)

This is the method using which anyone can query comm about the SIP security settings it applies on calls

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
value	*unsigned int*	This is the struct that has the relevant SIP security level fields comm is interested in

SECURITY_AUTO=0	for SECURITY_AUTO
SECURITY_OFF=1	for SECURITY_OFF
SECURITY_STRICT=2	for SECURITY_STRICT

EVENT sipSecurityPropertyModified(SecurityType value)

This is an even that comm throws when the SIP security settings it applies on calls are modified

Parameters

value	*unsigned int*	This is the struct that has the relevant SIP security level fields comm is interested in

SECURITY_AUTO=0	for SECURITY_AUTO
SECURITY_OFF=1	for SECURITY_OFF
SECURITY_STRICT=2	for SECURITY_STRICT

SYNC setH323SecurityProperty(const SecurityType value)

This is the method using which comm can be influenced to apply H323 security settings on the calls

privilege level ADMIN

Parameters

Inputs

value	*unsigned int*	This is the struct that has the relevant H.323 security level fields comm is interested in

SECURITY_AUTO=0	for SECURITY_AUTO
SECURITY_OFF=1	for SECURITY_OFF
SECURITY_STRICT=2	for SECURITY_STRICT

Outputs

_rv	*integer*	commReturnVal

COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
COMMRPC_FAILURE=-601	COMMRPC_FAILURE
COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getH323SecurityProperty(SecurityType *value)

This is the method using which anyone can query comm about the H323 security settings it applies on calls

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
value	*unsigned int*	This is the struct that has the relevant H.323 security level fields comm is interested in

SECURITY_AUTO=0	for SECURITY_AUTO
SECURITY_OFF=1	for SECURITY_OFF
SECURITY_STRICT=2	for SECURITY_STRICT

EVENT h323SecurityPropertyModified(SecurityType value)

This is an even that comm throws when the H323 security settings it applies on calls are modified

Parameters

value	*unsigned int*	This is the struct that has the relevant H.323 security level fields comm is interested in
		SECURITY_AUTO=0 for SECURITY_AUTO
		SECURITY_OFF=1 for SECURITY_OFF
		SECURITY_STRICT=2 for SECURITY_STRICT

SYNC setAutoBwFeature(int enable)

This is used to ask comm to enable/disable the autoBW feature

privilege level ADMIN

Parameters

Inputs

enable	*integer*	If autoBW feature needs to be enabled
		disabled=0 autoBW is disabled
		enabled=1 autoBW is enabled

Outputs

_rv	*integer*	commReturnVal
		COMMRPC_SUCCESS=0 COMMRPC_SUCCESS
		COMMRPC_FAILURE=-601 COMMRPC_FAILURE
		COMMRPC_FAILURE_INVALIDPARAM=-602 COMMRPC_FAILURE_INVALIDPARAM
		COMMRPC_FAILURE_NOTALLOWED_INCALL=-606 COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getAutoBwFeature(int *pEnable)

This is used to query comm about whether AutoBw feature is currently enabled or not.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pEnable	*integer*	If autoBW feature needs to be enabled
		disabled=0 autoBW is disabled
		enabled=1 autoBW is enabled

EVENT autoBwFeatureUpdated(int enabled)

This is an event that is thrown by comm when the AutoBW feature is enabled/disabled

Parameters

enabled	*integer*	If autoBW feature needs to be enabled
		disabled=0 autoBW is disabled
		enabled=1 autoBW is enabled

SYNC setStaticNATDetails(const StaticNATDetails *pDetails)

This is used to enable the Static NAT configuration

privilege level ADMIN

Parameters

Inputs

pDetails	*structure*	This is used to set the static NAT details
		bIsEnabled disabled=0 StaticNAT is disabled

enabled=1 StaticNAT is enabled

address[MAX_HOSTNAME_LEN] NULL terminated string indicating the static NAT IP address

Outputs

_rv *integer* commReturnVal

COMMRPC_SUCCESS=0 COMMRPC_SUCCESS

COMMRPC_FAILURE=-601 COMMRPC_FAILURE

COMMRPC_FAILURE_INVALIDPARAM=-602 COMMRPC_FAILURE_INVALIDPARAM

COMMRPC_FAILURE_NOTALLOWED_INCALL=-606 COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getStaticNATDetails(StaticNATDetails *pDetails)

This is invoked to query comm about enabling and details of static NAT configuration

privilege level ADMIN

Parameters

Outputs

_rv *integer* Return status (0 = success)

pDetails *structure* This is used to set the static NAT details

bIsEnabled

disabled=0 StaticNAT is disabled

enabled=1 StaticNAT is enabled

address[MAX_HOSTNAME_LEN] NULL terminated string indicating the static NAT IP address

EVENT staticNATDetailsUpdated()

This event is thrown when the static NAT configuration is modified

SYNC setStreamingRecordingDetails(const StreamRecrdngDetails *pDetails)

privilege level ADMIN

Parameters

Inputs

pDetails *structure* This struct contains the Recording specific details

bIsStreamingEnabled That is 1 - enable and 0 - disable

disabled=0 streaming is disabled

enabled=1 streaming is enabled

recorderHostName string recorder Host Name or address

recorderPort int recorder Port, by default 443. It allows minimum 1 digit and maximum 5 digit number (only digits). In case try to save without any value it uses default 443

recorderKey int recorder Key. It allows minimum 0 digit (empty) and maximum 10 digit number (only digits) and it is secure (It show asterisks instead of the numbers)

eRecordingLayout

RECORDING_LAYOUT_ALL_CALLERS=0 RECORDING_LAYOUT_ALL_CALLERS=0

RECORDING_LAYOUT_NEARVIDEO_ONLY=1 RECORDING_LAYOUT_NEARVIDEO_ONLY

RECORDING_LAYOUT_FARVIDEO_ONLY=2 RECORDING_LAYOUT_FARVIDEO_ONLY

Outputs

_rv *integer* commReturnVal

COMMRPC_SUCCESS=0 COMMRPC_SUCCESS

COMMRPC_FAILURE=-601 COMMRPC_FAILURE

COMMRPC_FAILURE_INVALIDPARAM=-602 COMMRPC_FAILURE_INVALIDPARAM

COMMRPC_FAILURE_NOTALLOWED_INCALL=-606 COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getStreamingRecordingDetails(StreamRecrdngDetails *pDetails)

This is the method using which anyone can query comm for the current applicable settings for streaming/recording

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pDetails	*structure*	This struct contains the Recording specific details
	bIsStreamingEnabled	That is 1 - enable and 0 - disable
		disabled=0 streaming is disabled
		enabled=1 streaming is enabled
	recorderHostName	string recorder Host Name or address
	recorderPort	int recorder Port, by default 443. It allows minimum 1 digit and maximum 5 digit number (only digits). In case try to save without any value it uses default 443
	recorderKey	int recorder Key. It allows minimum 0 digit (empty) and maximum 10 digit number (only digits) and it is secure (It show asterisks instead of the numbers)
	eRecordingLayout	RECORDING_LAYOUT_ALL_CALLERS=0 RECORDING_LAYOUT_ALL_CALLERS=0
		RECORDING_LAYOUT_NEARVIDEO_ONLY=1 RECORDING_LAYOUT_NEARVIDEO_ONLY
		RECORDING_LAYOUT_FARVIDEO_ONLY=2 RECORDING_LAYOUT_FARVIDEO_ONLY

EVENT streamingRecordingDetailsModified()

Comm throws this to indicate that some details related to streaming/recording have got modified

EVENT recordingPossible(int iRecordingPossible)

This is an event thrown by comm to indicate that it is possible to record a particular conference Comm might have certain situations mostly related to resource crunch when it is unable to perform recording at any stage of the conference. Then it can generate this event to indicate that recording is possible or not. Nothing needs to be assumed till this event is generated

Parameters

iRecordingPossible	*integer*	if recording this conference is possible
		disabled=0 recording is disabled
		enabled=1 recording is enabled

SYNC getRecordingPossible(int* iRecordingPossible)

This is the method that can be invoked on comm to ask whether recording is possible or not.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
iRecordingPossible	*integer*	if recording this conference is possible
		disabled=0 recording is disabled
		enabled=1 recording is enabled

ASYNC startRecording(unsigned int recordingPIN)

This is the method that can be invoked on comm to ask it to start recording the current conferenc

privilege level ADMIN

Parameters

Inputs

recordingPIN	*unsigned int*	This is PIN for recording. If this is a non-zero value, this PIN will override pre-configured pin

RESPONSE startRecordingDone(RecordingState eRetVal)

This is the asynch response sent by comm to indicate state of startrecording request

Parameters

`eRetVal` *unsigned int* This is the enum that conveys the response to the startRecording

RECORDING_STOPPED_SUCCESFULLY=0	RECORDING_STOPPED_SUCCESFULLY
RECORDING_STARTED_SUCCESFULLY=1	RECORDING_STARTED_SUCCESFULLY
RECORDING_START_INITIATED=2	RECORDING_START_INITIATED
RECORDING_STOP_INITIATED=3	RECORDING_STOP_INITIATED
RECORDING_START_FAILURE_NO_CONFERENCE=4	RECORDING_START_FAILURE_NO_CONFERENCE
RECORDING_START_FAILURE_NO_RESOURCE=5	RECORDING_START_FAILURE_NO_RESOURCE
RECORDING_START_FAILURE_VC_NOT_RESPONDING=6	RECORDING_START_FAILURE_VC_NOT_RESPONDING
RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED=7	RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED
RECORDING_START_FAILURE_GENERIC=8	RECORDING_START_FAILURE_GENERIC
RECORDING_STOP_FAILURE_GENERIC=9	RECORDING_STOP_FAILURE_GENERIC
RECORDING_START_FAILURE_INVALID_PIN=10	RECORDING_START_FAILURE_INVALID_PIN

`ASYNC stopRecording()`

This is the method that can be invoked on comm to ask it to stop recording the current conference being recorded

privilege level ADMIN

`RESPONSE stopRecordingDone(RecordingState eRetVal)`

This is the asynch response sent by comm to indicate state of stoprecording request

Parameters

`eRetVal` *unsigned int* This is the enum that conveys the response to the stopRecording

RECORDING_STOPPED_SUCCESFULLY=0	RECORDING_STOPPED_SUCCESFULLY
RECORDING_STARTED_SUCCESFULLY=1	RECORDING_STARTED_SUCCESFULLY
RECORDING_START_INITIATED=2	RECORDING_START_INITIATED
RECORDING_STOP_INITIATED=3	RECORDING_STOP_INITIATED
RECORDING_START_FAILURE_NO_CONFERENCE=4	RECORDING_START_FAILURE_NO_CONFERENCE
RECORDING_START_FAILURE_NO_RESOURCE=5	RECORDING_START_FAILURE_NO_RESOURCE
RECORDING_START_FAILURE_VC_NOT_RESPONDING=6	RECORDING_START_FAILURE_VC_NOT_RESPONDING
RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED=7	RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED
RECORDING_START_FAILURE_GENERIC=8	RECORDING_START_FAILURE_GENERIC
RECORDING_STOP_FAILURE_GENERIC=9	RECORDING_STOP_FAILURE_GENERIC
RECORDING_START_FAILURE_INVALID_PIN=10	RECORDING_START_FAILURE_INVALID_PIN

`SYNC getRecordingState(RecordingState* peRecordingState)`

This is the method that can be invoked on comm to ask the current recording status.

privilege level ADMIN

Parameters

Outputs

`_rv` *integer* Return status (0 = success)

`peRecordingState` *unsigned int* This is the enum that conveys current recording state

RECORDING_STOPPED_SUCCESFULLY=0	RECORDING_STOPPED_SUCCESFULLY
RECORDING_STARTED_SUCCESFULLY=1	RECORDING_STARTED_SUCCESFULLY
RECORDING_START_INITIATED=2	RECORDING_START_INITIATED
RECORDING_STOP_INITIATED=3	RECORDING_STOP_INITIATED
RECORDING_START_FAILURE_NO_CONFERENCE=4	RECORDING_START_FAILURE_NO_CONFERENCE
RECORDING_START_FAILURE_NO_RESOURCE=5	RECORDING_START_FAILURE_NO_RESOURCE

RECORDING_START_FAILURE_VC_NOT_RESPONDING=6	RECORDING_START_FAILURE_VC_NOT_RESPONDING
RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED=7	RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED
RECORDING_START_FAILURE_GENERIC=8	RECORDING_START_FAILURE_GENERIC
RECORDING_STOP_FAILURE_GENERIC=9	RECORDING_STOP_FAILURE_GENERIC
RECORDING_START_FAILURE_INVALID_PIN=10	RECORDING_START_FAILURE_INVALID_PIN

EVENT recordingStateChanged(RecordingState eRecordingState)

This is an event thrown by comm to indicate the change in recording state.

Parameters

eRecordingState *unsigned int* This is the enum that conveys current recording state.

RECORDING_STOPPED_SUCCESFULLY=0	RECORDING_STOPPED_SUCCESFULLY
RECORDING_STARTED_SUCCESFULLY=1	RECORDING_STARTED_SUCCESFULLY
RECORDING_START_INITIATED=2	RECORDING_START_INITIATED
RECORDING_STOP_INITIATED=3	RECORDING_STOP_INITIATED
RECORDING_START_FAILURE_NO_CONFERENCE=4	RECORDING_START_FAILURE_NO_CONFERENCE
RECORDING_START_FAILURE_NO_RESOURCE=5	RECORDING_START_FAILURE_NO_RESOURCE
RECORDING_START_FAILURE_VC_NOT_RESPONDING=6	RECORDING_START_FAILURE_VC_NOT_RESPONDING
RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED=7	RECORDING_START_FAILURE_RECORDING_ALREADY_STARTED
RECORDING_START_FAILURE_GENERIC=8	RECORDING_START_FAILURE_GENERIC
RECORDING_STOP_FAILURE_GENERIC=9	RECORDING_STOP_FAILURE_GENERIC
RECORDING_START_FAILURE_INVALID_PIN=10	RECORDING_START_FAILURE_INVALID_PIN

SYNC setSystemIdentification(const SystemIdentification *pSystemID)

This is the API using which comm can be informed about a newer value for the system identity.

privilege level ADMIN

Parameters

Inputs

pSystemID *structure* This is typically added in displayName for SIP/323 messages

Outputs

_rv *integer* commReturnVal

COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
COMMRPC_FAILURE=-601	COMMRPC_FAILURE
COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getSystemIdentification(SystemIdentification *pSystemID)

This is the API using which comm can be queried about the latest value for the system identity.

privilege level ADMIN

Parameters

Outputs

_rv *integer* Return status (0 = success)

pSystemID *structure* This is typically added in displayName for SIP/323 messages

EVENT systemIdentificationUpdated()

This is an event thrown by comm when the System Identification string is updated

SYNC setAMCFeature(int enable)

This is the API using which comm can be informed that AMC feature needs to be enabled/disabled

privilege level ADMIN

Parameters

Inputs

enable	*integer*	if AMC is enabled	
		disabled=0	AMC is disabled
		enabled=1	AMC is enabled

Outputs

_rv	*integer*	commReturnVal	
		COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
		COMMRPC_FAILURE=-601	COMMRPC_FAILURE
		COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
		COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getAMCFeature(int *pEnable)

This is the API using which anyone can query comm about AMC feature applicability

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pEnable	*integer*	if AMC is enabled
		disabled=0 AMC is disabled
		enabled=1 AMC is enabled

EVENT amcFeatureUpdated(int enable)

This is an event that is thrown by comm when the AMC feature is enabled/disabled

Parameters

enable	*integer*	if AMC is enabled	
		disabled=0	AMC is disabled
		enabled=1	AMC is enabled

SYNC setMaxBitrate(int maxBitrate)

This is used to set maximum bitrate value for the conference.

privilege level ADMIN

Parameters

Inputs

maxBitrate	*integer*	Bitrate in bytes and It should allow to set "Auto, 6000 kbps, 5000 kbps, 4000 kbps, 3500 kbps, 3000 kbps, 2500 kbps, 2000 kbps, 1536 kbps, 1408 kbps, 1280 kbps, 1152 kbps, 1024 kbps, 896 kbps, 768 kbps, 640 kbps, 512 kbps, 384 kbps, 320 kbps, 256 kbps, 192 kbps, 128 kbps" AUTO as the bandwidth, WEB UI need to set the value as 0 for setMaxBitrate. COMM will automatically figure out the meaning of AUTO and set the correct value. Auto means maximum bit rate supported by the system. This is 6 MB on sequoia

Outputs

_rv	*integer*	commReturnVal	
		COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
		COMMRPC_FAILURE=-601	COMMRPC_FAILURE
		COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
		COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getMaxBitrate(int* pMaxBitrate)

This is used to get the what maximum bitrate value for the conference.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
pMaxBitrate	*integer*	Bitrate in bytes

EVENT maxBitrateUpdated(int maxBitrate)

This is an event that is thrown by comm when Max Bitrate Value is updated

Parameters

maxBitrate	*integer*	maxbitrate in bytes

SYNC setBWSlider(int ancAllocation)

This API is used to set percentage of bit-rate to allocate for ancillary channel.

privilege level ADMIN

Parameters

Inputs

ancAllocation	*integer*	Percentage of bit-rate to allocate for ancillary channel. Valid values are from 10 to 50 with increment of 10

Outputs

_rv	*integer*	commReturnVal

COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
COMMRPC_FAILURE=-601	COMMRPC_FAILURE
COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getBWSlider(int* ancAllocation)

This API is used to get percentage of bit-rate being allocated for ancillary channel.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
ancAllocation	*integer*	Percentage of bit-rate to allocate for ancillary channel. Valid values are from 10 to 90 with increment of 10

EVENT bwSliderUpdated(int ancAllocation)

This is an event that is thrown by comm when BW slider value is updated

Parameters

ancAllocation	*integer*	percentage of bit-rate being allocated for ancillary channel.

SYNC setAutoCallBitRate(AutoCallBitRate eBitRate)

This is used to set system's auto call bit rate.

privilege level ADMIN

Parameters

Inputs

eBitRate	*unsigned int*	AUTO_CALL_BITRATE_768=0	768kbps needed for 720p30

AUTO_CALL_BITRATE_1152=1	1152kbps needed for 720p60
AUTO_CALL_BITRATE_1728=2	1728kbps needed for 1080p30
AUTO_CALL_BITRATE_2500=3	2500kbps needed for 1080p60
AUTO_CALL_BITRATE_INVALID=4	invalid

Outputs

_rv	*integer*	commReturnVal	
		COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
		COMMRPC_FAILURE=-601	COMMRPC_FAILURE
		COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
		COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getAutoCallBitRate(AutoCallBitRate* peBitRate)

This is used to get system's auto call bitrate.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
peBitRate	*unsigned int*	AUTO_CALL_BITRATE_768=0 — 768kbps needed for 720p30
		AUTO_CALL_BITRATE_1152=1 — 1152kbps needed for 720p60
		AUTO_CALL_BITRATE_1728=2 — 1728kbps needed for 1080p30
		AUTO_CALL_BITRATE_2500=3 — 2500kbps needed for 1080p60
		AUTO_CALL_BITRATE_INVALID=4 — invalid

SYNC setReservedPortRange(unsigned short lowerMediaPort, unsigned short upperMediaPort)

This API is used to set dynamic UDP/TCP port range for media, system will reboot on change. Range should be atleast 100.

privilege level ADMIN

Parameters

Inputs

lowerMediaPort	*unsigned int*	Lowest number in the reserved range of the media port number, valid value is between 49152 to 65535
upperMediaPort	*unsigned int*	Highest number in the reserved range of the media port number, valid value is between 49152 to 65535

Outputs

_rv	*integer*	commReturnVal	
		COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
		COMMRPC_FAILURE=-601	COMMRPC_FAILURE
		COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
		COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL

SYNC getReservedPortRange(unsigned short *lowerMediaPort, unsigned short *upperMediaPort)

This API is used to get dynamic UDP/TCP port range for media.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	commReturnVal	
		COMMRPC_SUCCESS=0	COMMRPC_SUCCESS
		COMMRPC_FAILURE=-601	COMMRPC_FAILURE
		COMMRPC_FAILURE_INVALIDPARAM=-602	COMMRPC_FAILURE_INVALIDPARAM
		COMMRPC_FAILURE_NOTALLOWED_INCALL=-606	COMMRPC_FAILURE_NOTALLOWED_INCALL
lowerMediaPort	*unsigned int*	Lowest number in the reserved range of the media port number, valid value is between 49152 to 65535	

`upperMediaPort` *unsigned int* Highest number in the reserved range of the media port number, valid value is between 49152 to 65535

EVENT portRangeUpdated()

This event is thrown when port range is updated

SYNC setLogLevel(int mask)

This API is used to set LogLevel

privilege level ADMIN

Parameters

Inputs

`mask`	*integer*	LogLevel mask value	
		LSLogVerboseDebug=0x01ff	Verbose and below level.
		LSLogDebug=0x00ff	Debug and below level.
		LSLogInfo=0x007f	Info and below level.
		LSLogWarning=0x003f	Warning and below level.
		LSLogError=0x001f	Error and below level.
		LSLogFailure=0x000f	Failure and below level.
		LSLogCritical=0x0007	Critical and below level.
		LSLogAlert=0x0003	Alert and below level.
		LSLogPanic=0x0001	Panic level only.
		LSLogDisable=0x0000	disable

Outputs

`_rv`	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

This API is used to invoke LogLevel value

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)	
`mask`	*integer*	LogLevel mask value	
		LSLogVerboseDebug=0x01ff	Verbose and below level.
		LSLogDebug=0x00ff	Debug and below level.
		LSLogInfo=0x007f	Info and below level.
		LSLogWarning=0x003f	Warning and below level.
		LSLogError=0x001f	Error and below level.
		LSLogFailure=0x000f	Failure and below level.
		LSLogCritical=0x0007	Critical and below level.
		LSLogAlert=0x0003	Alert and below level.
		LSLogPanic=0x0001	Panic level only.
		LSLogDisable=0x0000	disable

EVENT logLevelChanged(int mask)

This event is thrown when log Level Changes

Parameters

`mask`	*integer*	LogLevel mask value	
		LSLogVerboseDebug=0x01ff	Verbose and below level.
		LSLogDebug=0x00ff	Debug and below level.

LSLogInfo=0x007f	Info and below level.
LSLogWarning=0x003f	Warning and below level.
LSLogError=0x001f	Error and below level.
LSLogFailure=0x000f	Failure and below level.
LSLogCritical=0x0007	Critical and below level.
LSLogAlert=0x0003	Alert and below level.
LSLogPanic=0x0001	Panic level only.
LSLogDisable=0x0000	disable

EVENT customCommandExecuted(StringZ commandResult)

Event which is thrown by Comm when custom command execution is successful.

Parameters

`commandResult`	*string*	Command result

EVENT webProxyDetailsUpdated(int bIsWebProxyEnabled)

This is the event that is sent by comm to indicate that Web Proxy details have been modified

Parameters

`bIsWebProxyEnabled`	*integer*	Whether Web Proxy enabled or disabled, 1-Enable and 0-Disable

EVENT preferenceForProtocolUpdated()

event is fired when preference proto is updated

EVENT rvLogEnableChanged(int enable)

Parameters

`enable`	*integer*	If enable=1, enable RV logs, if enable=0, disable RV logs

EVENT RFC1918VerificationStatusUpdated()

This event is thrown when setRFC1918VerificationStatus status changes

EVENT InbandDtmfEnableUpdated()

This event is thrown when InbandDtmf status changes

EVENT bsModeSiren14Updated(BSMode eMode)

This event is thrown when setBSModeEnable status changes

Parameters

`eMode`	*unsigned int*	This is Byte swapped mode for Siren-14 audio 0 BSMODE_DEFAULT Default behavior, depends on the User agent info it will do byteswapped ps14 1 BSMODE_ENABLE Enable, Always do byteswapped ps14 2 BSMODE_DISABLE Disable, byteswapped ps14 turned off completely

EVENT allow2ndAudioCallUpdated(int enable)

This event is thrown when allow-2nd-audio-callvalue is changed.

Parameters

`enable`	*integer*	If enable = 1 2nd audio call is allowed, If enable = 0 - it is not

EVENT h264PM1EnableUpdated(int enable)

This event is thrown when H264PM1Enable value is changed.

Parameters

`enable` *integer* If enable = 1 packetization mode=1 is advertized, If enable = 0 - it is not

EVENT sipGenSupportValuesUpdated()

This is the event that is sent by comm to indicate that SIP general support configuration details have been modified

EVENT sipSupportValuesUpdated(SipContextType eContextType)

This is the event that is sent by comm to indicate that SIP support configuration details have been modified

Parameters

`eContextType` *unsigned int* This decides whether this refers to the primary or the secondary SIP configuration

- 0 Primary
- 1 Secondary

EVENT ConnectionsServerURLUpdated()

Connections Server URL has been updated

EVENT getConnectionsDetailsDone(StringZ connectionsData)

returns the connections data in CSV format string

Parameters

`connectionsData` *string* encapsulates the downloaded connections information

EVENT mediaEncryptionForConnectionsUpdated(int encyptMedia)

event which is thrown when MediaEncryption (Enabled/Disabled) value is changed for connections

Parameters

`encyptMedia` *integer* - 0 disable, 1 enable

EVENT signalingModeForConnectionsUpdated(CnxSignalingMode mode)

event which is thrown when Signaling Mode value is changed for connections

Parameters

`mode` *unsigned int* indicates the mode in which rtclient should connect to the rtserver for signaling channel

- 0 CNX_SIGNALING_MODE_DIRECT - Bypass rtclient and connect directly
- 1 CNX_SIGNALING_MODE_UDP - Connect using UDP
- 2 CNX_SIGNALING_MODE_TCP - Connect using TCP
- 3 CNX_SIGNALING_MODE_AUTO - Let rtclient decide the best mode
- 4 CNX_SIGNALING_MODE_TLS - Connect using TLS
- 5 CNX_SIGNALING_MODE_TUNNELED - Connect in tunneled mode
- 6 CNX_SIGNALING_MODE_ENCYPTED - Encrypted tunneled mode
- 7 CNX_SIGNALING_MODE_NONENCRYPTED - Unencrypted mode

EVENT h323SupportValuesUpdated()

This is the event that is sent by comm to indicate that H323 support configuration details have been modified

EVENT setupTransportAddrStatusUpdated()

This event is thrown when setupTransportAddrStatus changes

EVENT symmetricRtpUpdated()

This event is thrown when Symmetric Rtp Configuration changes

EVENT mediaDisconnectTimerUpdated()

This event is thrown when mediaDisconnectTimerCfg changes

EVENT noMediaDisconnectUpdated()

This event is thrown when noMediaDisconnect feature value changes

EVENT H323TcpKeepaliveUpdated()

This event is thrown when VSAT mode value changes

EVENT H241MaxStaticMbpsCfgUpdated()

CommScriptRunner Class Documentation

Overview

Functions by Group

CommScripts

commRebooted

GeneralSupportPage

factoryResetDone

EVENT commRebooted()

This is an event thrown to indicate that the comm has been rebooted

EVENT factoryResetDone()

Comm specific factory reset is successful

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

CommStats Class Documentation

Overview

Functions by Group

Other Functions

packetLossExperienced

EVENT packetLossExperienced(unsigned int commCallId,PacketLossLevel pktLossLevel)

This is an event that is thrown when we experience packet loss.

Parameters

commCallId	*unsigned int*	This is how comm refers to the call.
pktLossLevel	*unsigned int*	This is the level of packet loss that is being experienced

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

Conf Class Documentation

Overview

Functions by Group

Other Functions

isVersionCompatible_1_0

Conference and Call State

getNumConnectedCalls
getConferenceCallsByNatural
getConferenceCallsByTalker
getConferenceState
conferenceStateChanged
getCallDetails
callDetailsChanged
getCallState
callStateChanged
callFailed
callTransferred
answer
reject
terminate
userAckFailedCall
failedCallAcked
terminateConferenceCalls

Conference and Call Dialing

dialCallDefault
dialNumber
dialCall
dialCallDone
transferCall
transferCallDone

System State

getSystemDoNotDisturb
setSystemDoNotDisturb
systemDoNotDisturbChanged

Preferences

getAutoAnswer
setAutoAnswer
autoAnswerChanged
getAutoAnswerMute
setAutoAnswerMute
autoAnswerMuteChanged
getLogLevel
setLogLevel
logLevelChanged

SYNC isVersionCompatible_1_0()

determines if the current runtime is compatible with Version 1.0.x. use the WebServices.hasMethod() to test for compatibility

privilege level USER

Parameters

Outputs

_rv	*integer*	0 if not compatible, 1 if compatible

SYNC getNumConnectedCalls(int* total, int* video, int* voice)

Returns the number of connected calls of the entire system

privilege level USER

Parameters

Outputs

_rv	*integer*	>=0 if successful, as the total number of calls -ENOMEM if all output parameters were NULL <0 failed to connect or other errors
total	*integer*	The total number of calls (video + voice).
video	*integer*	The number of video calls.
voice	*integer*	The number of voice calls.

SYNC getConferenceCallsByNatural(ConstStringZ confUid, callids_t* calls)

Returns the call IDs in natural order. Natural order is the order the calls connected to the conference, with the first to connect in position 0.

privilege level ADMIN

Parameters

Inputs

confUid	*string*	The UID of the conference to query. Use an empty string for the default conference.

Outputs

_rv	*integer*	>=0 if successful as the total number of calls -ENOENT if conference is not found -ENOMEM if calls is NULL <0 failed to connect or other errors
calls	*structure*	All the call IDs in the conference. See callids_t for details.

SYNC getConferenceCallsByTalker(ConstStringZ confUid, callids_t* calls)

Returns the an array of call IDs in talker order. Talker order is the order the calls have talked, with the last talker in position 0.

privilege level ADMIN

Parameters

Inputs

confUid	*string*	The UID of the conference to query. Use an empty string for the default conference.

Outputs

_rv	*integer*	>=0 if successful as the total number of calls -ENOENT if conference is not found -ENOMEM if calls is NULL <0 failed to connect or other errors
calls	*structure*	All the call IDs in the conference. See callids_t for details.

SYNC getConferenceState(ConstStringZ confUid, confstate_t* state)

Returns the state of a conference.

privilege level USER

Parameters

Inputs

confUid	*string*	The UID of the conference to query. Use an empty string for the default conference.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found -ENOMEM if state is NULL <0 failed to connect or other errors
state	*structure*	The state of the conference. See confstate_t for details.

EVENT conferenceStateChanged(const confstate_t* state)

Posted when the state of a conference changes.

Parameters

state	*structure*	The state of the conference. See confstate_t for details.

SYNC getCallDetails(int callId, calldetails_t* details)

Returns the details of a call.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if call is not found <0 failed to connect or other errors
details	*structure*	The details of the call. See calldetails_t for details.

EVENT callDetailsChanged(const calldetails_t* details)

Posted when the details of a call change.

Parameters

details	*structure*	The details of the call. See calldetails_t for details.

SYNC getCallState(int callId, callstate_t* state)

Returns the state of a call.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if call is not found <0 failed to connect or other errors
state	*structure*	The state of the call. See callstate_t for details.

EVENT callStateChanged(const callstate_t* state)

Posted when the state of a call changes.

Parameters

state	*structure*	The state of the call. See callstate_t for details.

EVENT callFailed(const calldetails_t* details, const callstate_t* state, ConstStringZ error)

Posted when the a call terminates with an error condition

Parameters

details	*structure*	The details of the call prior to the termination. See calldetails_t for details.
state	*structure*	The state of the call prior to the termination. See callstate_t for details.
error	*string*	Any of the following error conditions busy: the callee was busy and could not accept the call unavailable: the callee system was unavailable to accept the call max-calls-exceeded: connecting this call would exceed the local resources

EVENT callTransferred(int oldCallId, int newCallId, ConstStringZ newName)

Posted when the call is transferred. Clients should update their data structures with the new callId and name.

Parameters

oldCallId	*integer*	The callId of the call transferred
newCallId	*integer*	The new callId after the transfer
newName	*string*	The new name of the far end after the transfer

ASYNC answer(int callId)

Answers the call

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The ID of the call to answer

ASYNC reject(int callId)

Rejects the call

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The ID of the call to answer

ASYNC terminate(int callId)

Terminates the call

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The ID of the call to terminate

ASYNC userAckFailedCall(int callId)

Acknowledges the failed call. Should be called by a UI after the user acknowledges the failure. The failedCallAcked() event is posted. Allows all instaneces of UIs to synchronize on user acknowledgement from one instanceo. It is best to not call automatically, but only in response to user acknoledgement.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The ID of the call to acknowledge. If 0==callId, all failed calls will be accepted.

EVENT failedCallAcked(int callId)

Posted when a failed call is acknowledged. The UI should remove its failed call dialog for this call.

Parameters

callId	*integer*	The ID of the acknowledged call. If 0==callId, all failed calls have been accepted.

SYNC terminateConferenceCalls(ConstStringZ confUid)

terminates all the calls in the specified conference

privilege level ADMIN

Parameters

Inputs

confUid	*string*	the UID of the conference to terminate. Use an empty string for the default conference.

Outputs

_rv	*integer*	>=0 if successful as the number of calls terminated -ENOENT if conference is not found <0 failed to connect or other errors

ASYNC dialCallDefault(ConstStringZ name, ConstStringZ number)

Add a call to the default conference with default parameters. type=video, protocol=auto, bitrate=0, from=api, transId=

privilege level ADMIN

Parameters

Inputs

name	*string*	the name of the system to call
number	*string*	the voice number, video number, or IP address of the system to call

ASYNC dialNumber(ConstStringZ number)

Add a call to the default conference with default parameters. name=number, type=video, protocol=auto, bitrate=0, from=api, transId=

privilege level ADMIN

Parameters

Inputs

number	*string*	the voice number, video number, or IP address of the system to call

ASYNC dialCall(ConstStringZ confUid, ConstStringZ name, ConstStringZ number, ConstStringZ type, ConstStringZ protocol, int bitrate, ConstStringZ from, ConstStringZ transId)

Add a call to the specified conference. The system attempts to dial the provided number with the given parameters

privilege level ADMIN

Parameters

Inputs

confUid	*string*	The UID of the conference to query. Use an empty string for the default conference.
name	*string*	the name of the system to call
number	*string*	the voice number, video number, or IP address of the system to call
type	*string*	the type of call to dial voice, video, empty string for conference default
protocol	*string*	the communications protocol to use when dialing the call. h323, sip, empty string for default
bitrate	*integer*	the bitrate for the call in kb/s, pass 0 to use the conference's value
from	*string*	where the call was dialed from in the system:

		agenda, directory, favorites, manual, meetings, recents, api, or empty string for default
transId	*string*	a value unique to the caller to identify a particular invocation. allows multiple calls to be added at once.

RESPONSE dialCallDone(int callId, int returnValue, ConstStringZ transId)

the response to dialCall. the event is posted when the dialing starts.

Parameters

callId	*integer*	the ID of the call
returnValue	*integer*	0 if successful -ENOENT if conference is not found -EAGAIN if the system is unavailable and cannot dial right now <0 other errors
transId	*string*	the transId used in the corresponding addCall invocation

ASYNC transferCall(int callId, ConstStringZ transConfUid)

transfer a call to a different conference

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the ID of the call to transfer
transConfUid	*string*	the UID of the conference to transfer the call into

RESPONSE transferCallDone(int callId, int returnValue)

the response to transferCall. the event is posted when the transfer is complete.

Parameters

callId	*integer*	the ID of the call
returnValue	*integer*	0 if successful -ENOENT if call or conference was not found -EINVAL if call was not an incoming call or already in target conference -EBUSY if call is already in the target conference <0 other errors

SYNC getSystemDoNotDisturb(int *value)

Returns the system do not disturb state. Incoming calls are rejected when enabled.

privilege level USER

Parameters

Outputs

_rv	*integer*	0==sucess, <= SB errors
value	*integer*	1==enabled, 0==disabled

SYNC setSystemDoNotDisturb(int value)

Returns the system do not disturb state. Incoming calls are rejected when enabled. The state is not preserved across system reboots.

privilege level ADMIN

Parameters

Inputs

value	*integer*	1==enabled, 0==disabled

Outputs

_rv	*integer*	0==success, <0 SB errors

EVENT systemDoNotDisturbChanged(int value)

Posted when the system do not disturb state changes

Parameters

value	*integer*	1==enabled, 0==disabled

SYNC getAutoAnswer(int* value)

Get the auto answer preference. Automatically answers incoming calls.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
value	*integer*	The auto answer setting. 1==enabled, 0==disabled

SYNC setAutoAnswer(int value)

Set the auto answer preference

privilege level ADMIN

Parameters

Inputs

`value`	*integer*	The new auto answer setting. 1==enabled, 0==disabled

Outputs

`_rv`	*integer*	0 if successful <0 failed to connect or other errors

`EVENT autoAnswerChanged( int value )`

Posted when the auto answer preference changes

Parameters

`value`	*integer*	The auto answer setting. 1==enabled, 0==disabled

`SYNC getAutoAnswerMute( int* value )`

Get the auto answer mute preference. Mutes the system if a call is automatically answered.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
`value`	*integer*	The auto answer mute setting. 1==enabled, 0==disabled

`SYNC setAutoAnswerMute( int value )`

Set the auto answer mute preference

privilege level ADMIN

Parameters

Inputs

`value`	*integer*	The new auto answer mute setting. 1==enabled, 0==disabled

Outputs

`_rv`	*integer*	0 if successful <0 failed to connect or other errors

`EVENT autoAnswerMuteChanged( int value )`

Posted when the auto answer mute preference changes

Parameters

value	*integer*	The auto answer setting. 1==enabled, 0==disabled

SYNC getLogLevel(int* mask)

Get the log level for the service

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
mask	*integer*	Returned mask

SYNC setLogLevel(int mask)

Set the log level for the service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	0 if successful <0 failed to connect or other errors

EVENT logLevelChanged(int mask)

Posted when the log level changes.

Parameters

mask	*integer*	returned mask

Data Class Documentation

Overview

Functions by Group

Setting the Log Level

setLogLevel
getLogLevel
logLevelChanged

File Access

openFile
deleteFile

Backup Services

startDBBackup
startDBBackupEncrypted
getDBBackupFilename
dbDataReady
startDBRestore
startDBRestoreEncrypted
dbRestoreComplete

Retrieving System Logs

getAllLogData
getCriticalLogData
logDataReady

Gather System Crash Data

startCoroner
coronerInfo
coronerProgress
coronerDataReady

Upgrading the System

getUpgradeInProgress
upgradeStarted
upgradeCheck
upgradeCheckDone
upgradeCheckDoneEvent
upgradeStart
upgradeFileOpen
upgradeProgress
upgradeProgressEvent
upgradeDone
upgradeDoneEvent

Diagnostics

SYNC setLogLevel(int mask)

Set the log level for the pref service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

returns the current log level mask of dataman

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
mask	*integer*	returned mask

EVENT logLevelChanged(int mask)

issued when the data manager log level changes

Parameters

mask	*integer*	new log level mask

SYNC openFile(ConstStringZ filename, int mode, int *filesize)

Open a file in /tmp/download or /tmp/upload

privilege level ADMIN

Parameters

Inputs

filename

	string	Name of file to open
mode	*integer*	Open mode
		0 Open for reading in /tmp/downloads
		1 Open for writing in /tmp/uploads
		2 Open for writing in /tmp/uploads and send progress back to client. Each progress indication is a 32 bit integet containing the amount received so far.

Outputs

_rv	*integer*	Socket for upload/download
filesize	*integer*	returns the number of bytes available for reading (when reading a file)

SYNC deleteFile(ConstStringZ filename, int updown)

Delete a file in the upload or download directory.

privilege level USER

Parameters

Inputs

filename	*string*	Name of file to delete
updown	*integer*	0 File is in download directory
		1 File is in upload directory

Outputs

_rv	*integer*	Return status (0 = success)

ASYNC startDBBackup()

Begin a DB backup, expect one dbDataReady event to indicate success or failure.

privilege level ADMIN

ASYNC startDBBackupEncrypted(ConstStringZ password)

Begin a DB backup, expect one dbDataReady event to indicate success or failure.

privilege level ADMIN

Parameters

Inputs

password	*string*	Password To be used for DES-3 encryption

SYNC getDBBackupFilename(StringZ filename, size_t filename_size, int* filesize, StringZ checkvalue, size_t

checkvalue_size)

Get the current backup filename (if any)

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status 0 success - EINVAL Filename is null or filename_size 0 or less negative Error code from scandir function
filename	*string*	Memory buffer to hold name of backup data file for use with Data_openFile. Value will be empty string if no backup file exist.
filesize	*integer*	Size of backup data file (if any), if null this value is not computed
checkvalue	*string*	Memory buffer to hold check value of backup data file (if any), if null this value is not computed

RESPONSE dbDataReady(ConstStringZ filename, size_t filesize, ConstStringZ checkvalue)

Response to startDBBackup

Parameters

filename	*string*	Name of data file to be gathered with Data_openFile function or an empty string value indicates the data service is not available and the original startDBBackup request has completed without generating an output file (do not call Data_openFile).
filesize	*unsigned int*	Number of bytes in data file.
checkvalue	*string*	check value of data file, if null this value was not computed

ASYNC startDBRestore(ConstStringZ filename, size_t filesize, ConstStringZ checkvalue)

Begin a DB restore, expect one dbDataReady event to indicate success or failure.

privilege level ADMIN

Parameters

Inputs

filename	*string*	Name of restore file provided to Data_openFile

filesize	*unsigned int*	Size of restore file in bytes provided to Data_openFile
checkvalue	*string*	check value of data file (if any), if null this value is not used

ASYNC startDBRestoreEncrypted(ConstStringZ filename, size_t filesize, ConstStringZ checkvalue, ConstStringZ password)

Begin a DB restore, expect one dbDataReady event to indicate success or failure.

privilege level ADMIN

Parameters

Inputs

filename	*string*	Name of restore file provided to Data_openFile
filesize	*unsigned int*	Size of restore file in bytes provided to Data_openFile
checkvalue	*string*	check value of data file (if any), if null this value is not used
password	*string*	Password to be used for decryption

RESPONSE dbRestoreComplete(int result, ConstStringZ filename, size_t filesize, ConstStringZ checkvalue)

Parameters

result	*integer*	0 on success or error code from the database restore function
filename	*string*	Name of restore file used
filesize	*unsigned int*	Size of restore file used
checkvalue	*string*	check value of data file, if null this value was not computed

ASYNC getAllLogData( )

Request that the system log data be made available via Data_openFile expect a logDataReady event to indicate the results. Any previously available system log will be deleted. Expect a file name that ends in 'syslog'

privilege level ADMIN

ASYNC getCriticalLogData( )

Request that the critical log data be made available via Data_openFile expect a logDataReady event to indicate the results. Any previously available critical error log will be deleted. Expect a file name that ends in 'errlog'

privilege level ADMIN

RESPONSE logDataReady(ConstStringZ filename, size_t filesize)

Response to getAllLogData

Parameters

filename	*string*	Name of generated log data file to be used with Data_openFile function or an empty string value indicates the log data service is not available and the original request has completed without generating an output file (do not call Data_openFile).
filesize	*unsigned int*	Number of bytes in generated log data file.

ASYNC startCoroner(StringZ autopsy_file, StringZ user_message)

Request new autopsy execution by coroner. Response will be either coronerInfo(success) or coronerDataReady(failure).

privilege level ADMIN

Parameters

Inputs

autopsy_file	*string*	location of an autopsy file to execute, empty string or "-" will use default file
user_message	*string*	a user specified text string to add to the autopsy report

RESPONSE coronerInfo(int step_number, StringZ step_name, StringZ step_description, int step_estimated_time)

Response to sucessful startCoroner request. coronerInfo indicates coroner has located and parsed autopsy file. All autopsy steps will be numbered and enumerated using this event before any steps are executed. This allows a client to learn the order of execution and estimated execution time before executing an autopsy. The next coronerProgress event will indicate enumeration has completed.

Parameters

step_number	*integer*	step number in autopsy
step_name	*string*	step name
step_description	*string*	step description, what it will do
step_estimated_time	*integer*	step time estimate, how long will it take in seconds

RESPONSE coronerProgress(int step_number, int type, StringZ step_progress, int step_time_remaining)

Response to sucessful startCoroner request, marks end of coronerInfo events.

Parameters

step_number	*integer*	step number in autopsy file
type	*integer*	Type of progress message 0 Success, unused today 1 Information, info that is user friendly 2 Warning, unused today 3 Fail (error continuing), unused today 4 Halt (error stopping), unused today
step_progress	*string*	text describing progress
step_time_remaining	*integer*	remaining time for this step in seconds

RESPONSE coronerDataReady(ConstStringZ filename, size_t size)

Final response to startCoroner request, no more coronerProgress or coronerInfo events will occur.

Parameters

filename	*string*	location of results from coroner operation or empty string if coroner failed.
size	*unsigned int*	size of all data collected by coroner operation, a 0 size indicates no data is available.

SYNC getUpgradeInProgress(int* upgradeInProgress)

Queries device to determine if an upgrade is in progress.

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
upgradeInProgress	*integer*	Current upgrade status 0 No upgrade is occuring nonzero an upgrade is in progress

EVENT upgradeStarted( )

Indicates an upgrade start request has been received and an upgrade cycle is in progress.

`ASYNC upgradeCheck( ConstStringZ url )`

Uses url to check upgrade information. Expect Data_upgradeCheckDone response. The check requires only the first 100*1024(100k) bytes of a release file.

privilege level ADMIN

Parameters

Inputs

`url`	*string*	URL to release file. If empty string then the default value is file://localhost/tmp/upload/check.bin (the file check.bin from Data_openFile) http://host/filename uses wget to retrieve release file.

`RESPONSE upgradeCheckDone( int overallResult, ConstCheckResults results )`

Response to Data_upgradeCheck.

Parameters

`overallResult`	*integer*	Final result -6 Failed, request rejected, with unknown result -5 Failed, request rejected, no thread data available -4 Failed, request rejected, no thread created -3 Failed, request rejected, exclusive access denied -2 Failed, request rejected, command failed -1 Release upgrade bad, see param results for details 1 Release ok, see param results for details
`results`	*structure*	Structure holding results from upgrade checking. Values of 0 or greater are ok while negative values indicate errors. fileCheck — File check status -2 File has bad format -1 File has bad signature

	0	File check not performed
	1	File is ok
versionCheck	Version check status	
	- 2	Fail, system cannot be downgraded to any version less than X
	- 1	Fail, wrong device
	0	Version check not performed
	1	OK, upgrade, incoming version is higher
	2	OK, downgrade, incoming version is lower
	3	OK, reinstall, incoming version is same
licenseCheck	License status	
	- 3	Fail, License is expired, disallow upgradede and exit
	- 2	Fail, New build date is invalid or nonexistent disallow upgrade and exit
	- 1	Fail, Serial number is invalid or nonexistent disallow upgrade and exit
	0	License check not performed
	1	OK, factory

		license applies
	2	OK, local license applies
	3	OK, remote license applies
	4	OK, license check ignored
typeCheck	Type check status	
	0	Type check not performed
	1	Production upgrade from production system
	2	Production upgrade from development system
	3	Development upgrade from development system
	4	Development upgrade from production system
defaultsCheck	Defaults check status	
	0	Defaults check not performed
	1	Values will be preserved
	2	Values must be restored to defaults for image to install
allowedCheck	Is upgrade allowed?	
	- 2	Upgrade not allowed upgrade in progress
	- 1	Upgrade not allowed calls in progress
	0	Allowed check not performed
	1	Upgrade allowed

restartCheck	Does this upgrade require a restart? 0 Restart check not performed 1 Restart will be performed 2 No restart needed

EVENT upgradeCheckDoneEvent(int overallResult, ConstCheckResults results)

Event that shadows the upgradeCheckDone response.

Parameters

overallResult	*integer*	See upgradeCheckDone
results	*structure*	See upgradeCheckDone

ASYNC upgradeStart(ConstStringZ url, int flags)

Uses url to upgrade device. Expect Data_upgradeProgress and Data_upgradeDone responses. Request is complete after Data_upgradeDone event is sent.

privilege level ADMIN

Parameters

Inputs

url	*string*	URL to release file. If empty string then the default value is file://localhost/tmp/upload/upgrade.bin (the file upgrade.bin from Data_openFile) http://host/filename uses wget to retrieve release file. socket+teetar://localhost opens a socket to receive the file, see Data_upgradeFileOpen to get the socket number opened
flags	*integer*	Combination of flag bits controlling the upgrade 0x0001 Reset to defaults occurs after upgrade 0x0002 Force upgrade if calls or upgrade in progress 0x0010 Send verbose progress messages

RESPONSE upgradeFileOpen(int socket)

Response generated after Data_upgradeStart is called when url is socket+teetar, indicates it is ok to begin sending the upgrade file on the socket number specified.

Parameters

socket	*integer*	The socket number to send an upgrade image to.

RESPONSE upgradeProgress(int step, int maxStep, ConstStringZ progressInfo)

These events are generated after Data_upgradeStart is called and will continue until Data_upgradeDone occurs.

Parameters

step	*integer*	current upgrade step number, 1 based
maxStep	*integer*	maximum number of steps, 1 based
progressInfo	*string*	A short string describing the current upgrade steps

started	Upgrade started
reading	Reading data from specified url
verifyingVerifying	contents and format of upgrade image
extracting	Extracting required files from upgrade image
checking	Checking version, license, other
newroot	Creating new root file system for install destination
extracting_files	Extracting new files
installing	Installing new files
uboot	Upgrading uboot system launch part
fpga	Upgrading fpga parts
sync	Closing and finalizing new root partition
active_part	Updating the active partition
defaults	Restore to defaults
reboot	Rebooting system
removing	Removing temporary files from device

EVENT upgradeProgressEvent(int step, int maxStep, ConstStringZ progressInfo)

Event that shadows the upgradeProgress response.

Parameters

step	*integer*	current upgrade step number, 1 based
maxStep	*integer*	maximum number of steps, 1 based
progressInfo	*string*	A short string describing the current upgrade steps (see upgradeProgress)

RESPONSE upgradeDone(int result)

Final response to Data_upgradeStart request, no more events will occur.

Parameters

result *integer* Final result of upgrade

-6	Failed, request rejected, with unknown result
-5	Failed, request rejected, no thread data available
-4	Failed, request rejected, no thread created
-3	Failed, request rejected, exclusive access denied
-2	Failed, request rejected, command failed
0	Success
1	Failure, upgrade in progress
2	Failure, release file is bad format
3	Failure, release file has bad signature
4	Failure, release file missing required file
5	Failure, release file has bad manifest
6	Failure, device serial number is not valid
7	Failure, bad build date
8	Failure, upgrade file is for a different device
9	Failure, downgrade to previous version not allowed
10	Failure, license expired
11	Failure, upgrade not allowed calls in progress
12	Failure, upgrade not allowed restore to defaults required but not requested
13	Failure, unexpected partition scheme
16	Failure, could not make room for incoming image
17	Failure, extracted file verification error

EVENT upgradeDoneEvent(int result)

Event that shadows the upgradeDone response.

Parameters

result	*integer*	Final result of upgrade (see upgradeDone)

SYNC startDiag(ConstStringZ script)

Start a diagnostic script

privilege level ADMIN

Parameters

Inputs

script	*string*	Name of script to run

Outputs

_rv	*integer*	Return status (0 = success)

SYNC stopDiag()

Abort a currently running diagnostic script.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)

SYNC doDiagStep(int step, int ofSteps, int result, ConstStringZ message)

Called by a diagnostic script to indicate progress.

privilege level ADMIN

Parameters

Inputs

step	*integer*	Current step
ofSteps	*integer*	Number of total steps expected
result	*integer*	Error result for the last step
message	*string*	Human readable message for last step.

Outputs

_rv	*integer*	Return status (0 = success)

EVENT diagStarted()

Issued when a diagnostic script is started.

EVENT diagStep(int step, int ofSteps, int result,

`ConstStringZ message)`

Issued on behalf of a diagnostic script to indicate progress.

Parameters

`step`	*integer*	Current step
`ofSteps`	*integer*	Number of total steps expected
`result`	*integer*	Error result for the last step
`message`	*string*	Human readable message for last step.

`EVENT diagDone(int result)`

Issued when a diagnostic script completes.

Parameters

`result`	*integer*	Overall result of the diagnostic script (exit status)

Directory Class Documentation

Overview

A directory service actor enables access to a database of directory entries. This can be a local database or a remote database. Each database must have a unique name which is used to register the database with the master directory service. This requires that the initialization of the directory server object use the name in the C++ constructor. Example:

```
Directory_server *srv = new Directory_server("skype_directory")
```

Every database is expected to maintain a locally unique id for each entry in the database. This id is always represented as a string within the context of the directory service but may be an integer or some other type in the actual database. Contact info is represented as a JSON structure with the following fields: "id", "firstname", "lastname", "system", "refid", "number", "type", "protocol", and "bandwidth". These fields may be mapped by the directory service to other fields in the actual remote database, or in some cases, synthesized from fields in the remote database (i.e. firstname and lastname could be derived from a single name field, or dial could be a combination of several fields).

Directory services should use the client methods described in "Managing Directory Services" to register with the master server. Only the master should implement the server methods described there.

Functions by Group

Querying the directory database

beginQuery
itemsRemaining
abortQuery
getNext
requestNext
queryResult

Editing the database

saveEntry
saveEntryDone
deleteEntry
deleteEntryDone
clearAllEntries
clearAllEntriesDone
setConfiguration
getConfiguration
configurationChanged
setLogLevel
getLogLevel
logLevelChanged

presenceChanged
entryCreated
entryUpdated
entryDeleted
reload

Managing directory services

registerServer
deleteServer
getServerEnumerator
enumerateNextServer
requestServers

SYNC beginQuery(ConstStringZ query)

Start a directory lookup operation. The query is a pattern string. The query should return all records where the pattern matches the first characters in any word of the fields "firstname", "lastname", or "systemname". Words are defined as contiguous sequences of alphanumeric characters, so a systemname of "CR-AUS-Alamo" should match the query "ala". Queries are case insensitive.

privilege level ADMIN

Parameters

Inputs

query	*string*	Query specification

Outputs

_rv	*integer*	Query handle or negative error code

SYNC itemsRemaining(int handle)

Get the number of result rows remaining in an active query. This may be an approximation, due to additional filtering needed on the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Number of result entries, or negative error code

ASYNC abortQuery(int handle)

Close out a query before fetching all of the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

SYNC getNext(int handle, StringZ buffer, size_t buffer_size)

Retrieve the next result group from a directory query synchronously. Interchangable with requestNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Result status
		0 Success
		-1 End of query
		-EAGAIN Query not ready, try again
		-EINVAL Bad query handle
		-E2BIG Buffer too small
buffer	*string*	Buffer to accept query results. Buffer will be formatted with a JSON array of structures representing entries. Each entry will have the fields requested in the query.

ASYNC requestNext(int handle)

Request the next result group from a directory query asynchronously. Interchangable with getNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

RESPONSE queryResult(int handle, int status, ConstStringZ result)

Response to requestNext containing query results.

Parameters

handle	*integer*	Query handle used in requestNext
status	*integer*	Query status
		0 Success
		-1 End of query

-EINVAL Bad query handle

result *string* A JSON array of structures representing directory entries matching the query.

ASYNC saveEntry(ConstStringZ info)

Create a new entry in the database or update an existing entry. This will normally only be possible for the local directory. If the entry contains an id field that matches an existing entry in the database then this will perform an update, otherwise it will create the entry.

privilege level ADMIN

Parameters

Inputs

info *string* Contact info record in JSON format

RESPONSE saveEntryDone(int result, ConstStringZ info)

Signals the end of a saveEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded
- -EPERM Operation not allowed
- -EINVAL Invalid info record

info *string* JSON data for contact with directory added id field if no id was present in the original entry.

ASYNC deleteEntry(ConstStringZ id)

Delete an entry in the database. This will normally only be possible for the local directory.

privilege level ADMIN

Parameters

Inputs

id *string* Id field of record to delete.

RESPONSE deleteEntryDone(int result)

Signals the end of a deleteEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded

-EPERM	Operation not allowed
-EINVAL	Invalid info record

ASYNC clearAllEntries()

Remove all entries from the directory.

privilege level ADMIN

RESPONSE clearAllEntriesDone()

Signals the end of a clear-all operation.

SYNC setConfiguration(ConstStringZ config)

Configure remote server parameters. The format of the configuration object can be different for each type of directory service. In general it may include a server address, username, password, and base search string. The local directory does not require this command.

privilege level ADMIN

Parameters

Inputs

config	*string*	JSON formatted configuration object

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getConfiguration(StringZ config, int config_size)

Retrieve the current configuration.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error
config	*string*	Buffer to receive current configuration

EVENT configurationChanged(ConstStringZ server, ConstStringZ config)

Issued when the configuration is changed by setConfiguration.

Parameters

server

	string	Name of reconfigured server
config	*string*	New configuration

SYNC setLogLevel(int mask)

Set the logging mask.

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New logging mask

Outputs

_rv	*integer*	0 or negative error code

SYNC getLogLevel(int *mask)

Retrieve the current logging mask

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error code.
mask	*integer*	logging mask

EVENT logLevelChanged(int mask)

Issued when the logging level is changed.

Parameters

mask	*integer*	New logging level

EVENT presenceChanged(ConstStringZ server, ConstStringZ entries)

Issued to indicate a change of presence for one or more entries.

Parameters

server	*string*	Directory service that provides this entry
entries	*string*	JSON array of entries changed

EVENT entryCreated(ConstStringZ server, ConstStringZ info)

issued when a new entry is created in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Created entry

EVENT entryUpdated(ConstStringZ server, ConstStringZ info)

issued when an entry is updated in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Updated entry

EVENT entryDeleted(ConstStringZ server, ConstStringZ info)

issued when an entry is deleted from the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Deleted entry

EVENT reload(ConstStringZ server)

issued when cached entries for this directory should be invalidated.

Parameters

server	*string*	Actor name for this directory LISTEN Directory_requestServers()

SYNC registerServer(ConstStringZ serverName)

Add a directory server to the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of new server

Outputs

_rv	*integer*	Result status

0	Success
negative	Negative error code

SYNC deleteServer(ConstStringZ serverName)

Remove a directory server from the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of server to remove.

Outputs

_rv	*integer*	Result status
	0	Success
	-ENOENT	Server name not found

SYNC getServerEnumerator()

Begin enumeration of directory servers.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextServer(int handle, StringZ buffer, size_t buffer_size)

Get the next set of servers in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getServerEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has one or more server names.
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive server names. Names are space seperated. As many names as will fit in the buffer are returned.

EVENT requestServers()

Sent by the master directory to request that all directory servers register their name.

Event Class Documentation

Overview

The Event Manager provides a way to have system events sent to an external URL.

Functions by Group

Managing Event Subscriptions

subscribe
cancel

Enumerating Subscriptions

enumerateSubscriptions
getNextSubscription
enumerateSubscribedEvents
getNextEvents

Change Events

subscriptionChanged

SYNC subscribe(ConstStringZ id, ConstStringZ url, ConstStringZ events, int timeout)

Create or modify an event subscription. If a subscription already exists with the id provided then then events in this request will be added to the existing subscription and the timeout of the subscription will be set to the new timeout value. If no subscription currently exists with this id then a new subscription is created. To refresh the timeout the events string should be empty. A subscriptionChanged event will immediately be sent to the URL to test the viability of the connection.

privilege level ADMIN

Parameters

Inputs

id	*string*	A unique string to identify this subscription. A UUID is suggested.
url	*string*	URL to be posted to when subscribed events occur.
events	*string*	A space or comma separated list of events to add to the subscription.

timeout	*integer*	Number of seconds until this subscription expires. A value of zero is considered infinite.

Outputs

_rv	*integer*	Return status (0 = success)

SYNC cancel(ConstStringZ id)

Cancel an event subscription.

privilege level ADMIN

Parameters

Inputs

id	*string*	The unique identifier of the event subscription. This subscription will be deleted.

Outputs

_rv	*integer*	Return status (0 = success)

SYNC enumerateSubscriptions()

Get a list of current event subscriptions

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC getNextSubscription(int handle, StringZ subscription, int subscription_size)

Get the next subscription in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle

Outputs

_rv	*integer*	Status 0 Success, Buffer contains valid subscription data - End of enumeration or Enumeration 1 not found. No valid data in buffer.
subscription	*string*	Buffer to hold subscription information. Each subscription is a JSON object containing the following structure

id	Subscription identifier
url	Subscription URL
nevents	Number of events subscribed
timeout	Time left till timeout
status	A status string with one of the forms: "pending", "ok ", or "fail " where is the time of the last contact attempt. If no contact has been attempted yet then the string will be "pending".

SYNC enumerateSubscribedEvents(ConstStringZ id)

Enumerate the events subscribed to by the designated subscription.

privilege level ADMIN

Parameters

Inputs

id	*string*	Subscription id.

Outputs

_rv	*integer*	Enumerator handle or negative error.

SYNC getNextEvents(int handle, StringZ events, int events_size)

Get the next set of events in a subscription enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle

Outputs

_rv	*integer*	Status 0 Success, Buffer contains valid event data - End of enumeration or Enumeration not 1 found. No valid data in buffer.
events	*string*	Buffer to receive JSON array of event names.

EVENT subscriptionChanged(ConstStringZ id, ConstStringZ status)

Issued when a subscription is created, modified, or cancelled.

Parameters

id	*string*	Subscription identifier

`status`	*string*	Change type	
		"created"	New subscription created
		"modified"	New events added to existing subscription
		"refresh"	Timeout reset
		"cancel"	Subscription cancelled

Fan Class Documentation

Overview

The Fan class provides information about the system cooling resources.

Functions by Group

Getting Fan Information

SYNC getNumberOfFans(int* min, int* max, int* count)

Get the fan numbering scheme for the device

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
min	*integer*	minimum fan number this interface accepts
max	*integer*	maximum fan number this interface accepts
count	*integer*	total count of fans this interface manages

SYNC getInfo(size_t fan, int* current, StringZ status, size_t status_size)

Get all fan info for specified fan number

privilege level USER

Parameters

Inputs

fan	*unsigned int*	the fan to query

Outputs

_rv	*integer*	Return status

0	Success	
-EINVAL	Current is NULL, status is NULL, or status has 0 size	
-ENODEV	Fan number specified is not valid	

current	*integer*	current tachometer reading in revolutions per minute
	normal	Indicates fan is operating normally
	stalled	Indicates fan has stalled
status	*string*	String describing this fan

SYNC getTach(size_t fan)

Get specified fan tach reading in revolutions per minute

privilege level USER

Parameters

Inputs

fan	*unsigned int*	the fan number to investigate see getNumberOfFans for min to max range

Outputs

_rv	*integer*	Fan tachometer reading in RPM
	positive	Fan tach value in RPM
	-1	Bad fan specified as param

SYNC getDriveSpeed(void)

Get drive speed being applied now

privilege level USER

Parameters

Outputs

_rv	*integer*	0-127	Fan drive speed value
		-1	Error accessing fan device

EVENT statusChanged(size_t fan, int tach, ConstStringZ status)

Indicates a fan status has changed

Parameters

fan	*unsigned int*	The fan number being reported
tach	*integer*	the fan tachometer in revolutions per minute

`status`	*string*	stalled	Indicates the fan is stalled
		normal	Indicates the fan is operating normally

Fips Class Documentation

Overview

The FIPS daemon manages FIPS operation mode.

Functions by Group

Getting and setting FIPS mode

getFipsMode
setFipsMode

SYNC getFipsMode(int *mode)

Retrieve current FIPS mode of operation

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
mode	*integer*	Returned mode

SYNC setFipsMode(int mode)

Enable or disable FIPS mode of operation

privilege level ADMIN

Parameters

Inputs

mode	*integer*	New mode
		0 Disable FIPS mode
		1 Enable FIPS mode

Outputs

_rv	*integer*	Return status (0 = success)

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

Gui Class Documentation

Overview

Functions by Group

General

isVersionCompatible_1_0

Visible State

sleepNow
sleeping
wakeNow
awake
isAwake
hideNow
hidden
showNow
shown
isShown
getVideoMute
setVideoMute
videoMuteChanged
showMessage
clearFailedCallDialog

Presentation Management

startPresentation
stopPresentation
getPresentationState
presentationStateChanged

Recording Session Management

startRecordingSession
stopRecordingSession
getCurrentRecordingSession
currentRecordingSessionChanged

Output Management

getPhysicalDisplayArrangement
setPhysicalDisplayArrangement
physicalDisplayArrangementChanged

Primary Input

nextCamera
prevCamera
switchCamera
getCurrentCameraIndex
currentCameraIndexChanged

Presentation Input

Input Utilities

Layout Management

Directory Management

Initial Configuration

Safe Area

Clock Format

Language

Admin Passcode

getAdminPasscode
setAdminPasscode
adminPasscodeChanged

Call Utility

getCallStartTime
callStartTimeChanged

Log Level

getLogLevel
setLogLevel
logLevelChanged

SYNC isVersionCompatible_1_0()

determines if the current runtime is compatible with Version 1.0.x. use the WebServices.hasMethod() to test for compatibility

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	1==compatible 0==not compatible

ASYNC sleepNow()

Puts the UI to sleep

privilege level ADMIN

EVENT sleeping()

Posted when the UI sleeps

ASYNC wakeNow()

Wakes the UI

privilege level ADMIN

EVENT awake()

Posted when the UI wakes from sleep

SYNC isAwake()

Returns the wake state of the UI

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	1==awake 0==sleeping <0 SB error

ASYNC hideNow()

Hides the UI.

privilege level ADMIN

EVENT hidden()

Posted when the UI becommes hidden (aka transparent).

ASYNC showNow()

Shows the UI.

privilege level ADMIN

EVENT shown()

Posted when the UI becommes shown.

SYNC isShown()

Returns the shown state of the UI.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	1==shown 0==hidden <0 SB error

SYNC getVideoMute(int *muted)

Gets the video mute state.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0==success <0 SB error
muted	*integer*	1==muted 0==unmuted

SYNC setVideoMute(int muted)

Sets the video mute state. When muted, the local video from the primary and presentation inputs is transmitted. The video mute state can only be changed when a video call is connected, and defaults to 0 when the last video call disconnects.

privilege level ADMIN

Parameters

Inputs

muted	*integer*	1==muted 0==unmuted

Outputs

_rv	*integer*	0==success <0 SB error

EVENT videoMuteChanged(int muted)

Posted when the video mute state changes

Parameters

muted	*integer*	1==muted 0==unmuted

ASYNC showMessage(ConstStringZ message)

Directs the UI to show the provided message in a dialog with an OK button Multiple calls to showMessage() only shows the last message.

See Qt Rich Text for HTML details.

privilege level ADMIN

Parameters

Inputs

message	*string*	A UTF-8 string. It can contain some simple HTML 3.2 tags supported by Qt

ASYNC clearFailedCallDialog()

Directs the UI to clear the failed call dialog if it is showing

privilege level ADMIN

ASYNC startPresentation()

Starts the presentation

privilege level ADMIN

ASYNC stopPresentation()

Stops the presentation

privilege level ADMIN

SYNC getPresentationState(StringZ state, size_t state_size)

Returns the current presentation state of the system

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM not enough room in state to store the current state <0 failed to connect or other errors
state	*string*	The presentation state of the system: local, tx, rx, empty

EVENT presentationStateChanged(ConstStringZ state)

Posted when the presentation state changes

Parameters

state	*string*	The presentation state of the system. See getPresentationState() for value details.

SYNC startRecordingSession(ConstStringZ type)

Start a new recording session. Once active use Comm_startRecording() and Comm_stopRecording() to start/stop recording. If you chose the dual-presentation session then use Gui_startPresentation() and Gui_stopPresentation() to start/stop the 2nd stream. If you chose dual-cameras, the 2nd stream is started automatically.

NOTE: calling Comm_startRecording() without starting a recording session automatically starts the dual-cameras or dual-presentation session as appropriate.

privilege level ADMIN

Parameters

Inputs

type	*string*	Recording session type: camera, dual-cameras, presentation, dual-presentation dual-cameras is only available when 2 cameras are connected presentation and dual-presentation are only available when only 1 camera is connected

Outputs

_rv	*integer*	0 if successful -EINVAL the type is not valid for current scenario <0 failed to connect or other errors

ASYNC stopRecordingSession()

Stops the current recording session if there is one.

privilege level ADMIN

SYNC getCurrentRecordingSession(StringZ value, size_t value_size)

Get the current recording session

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM not enough room in value to store the current recording session <0 failed to connect or other errors
value	*string*	The current recording session

EVENT currentRecordingSessionChanged(ConstStringZ value)

Posted when the recording session changes

Parameters

value	*string*	The new recording session. Black if no recording session active.

SYNC getPhysicalDisplayArrangement(StringZ value, size_t value_size)

Gets the physical arrangement of your displays attached to the codec. Provides a hint to the system to direct which content shows on the displays.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM not enough room in value to store the data <0 failed to connect or other errors
value	*string*	The current physical display arrangement. See setPhysicalDisplayArrangement for details about value.

SYNC setPhysicalDisplayArrangement(ConstStringZ value)

Sets the physical arrangement of your displays attached to the codec. Provides a hint to the system to direct which content shows on the displays.

privilege level ADMIN

Parameters

Inputs

value	*string*	The current physical display arrangement: apart — the displays are arranged physically apart in the room. only one display can been seen at a time. adjacent — the displays are arranged physically adjacent (together, side by side, top/bottom, etc.). both displays can been seen at the same time. mirrored — if the display arrangement is not interesting to you, but you want them to show the same content. single — only one display connected default — the original behavior, similar to apart.

Outputs

_rv

integer 0 if successful
-EINVAL no dual display license or not a valid value
<0 failed to connect or other errors

EVENT physicalDisplayArrangementChanged(ConstStringZ value)

Posted when the physicalDisplayArrangement changes

Parameters

value *string* The current physical room layout. See setPhysicalDisplayArrangement for details about value.

ASYNC nextCamera()

Switches to the next camera if more than one camera connected

privilege level ADMIN

ASYNC prevCamera()

Switches to the previous camera if more than one camera connected

privilege level ADMIN

ASYNC switchCamera(ConstStringZ camera)

Switches to the specified camera if more than one camera connected

privilege level ADMIN

Parameters

Inputs

camera *string* the camera to switch to: hdmi0 or dvi0

SYNC getCurrentCameraIndex(int* index)

Returns the index of the current camera

privilege level ADMIN

Parameters

Outputs

_rv *integer* Return status (0 = success)

index *integer* the index of the current camera

EVENT currentCameraIndexChanged(int index)

Posted when the index of the current camera changes

Parameters

index	*integer*	the index of the current camera

SYNC getPrimaryInput(StringZ value, size_t value_size)

Get the primary input

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
value	*string*	The primary input

EVENT primaryInputChanged(ConstStringZ value)

Posted when the primary input changes

Parameters

value	*string*	The primary input

SYNC getPresentationInput(StringZ value, size_t value_size)

Get the presentation input

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
value	*string*	The presentation input

EVENT presentationInputChanged(ConstStringZ value)

Posted when the presentation input changes

Parameters

value	*string*	The presentation input

SYNC getAvailableInputs(StringZ value, size_t value_size)

Get the available inputs

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
value	*string*	A space separated list of inputs

SYNC getInputHandle(ConstStringZ input, int *handle)

Returns the device handle for the priven input

privilege level ADMIN

Parameters

Inputs

input	*string*	The input to map to a handle

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
handle	*integer*	The handle of the input, -1 if the input was invalid

SYNC getAvailableLayouts(StringZ layouts, size_t layouts_size)

Returns the currently available layouts given the system state

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	>= 0 The number of layouts < 0 Normal SB failures
layouts	*string*	A space delimited list of layouts in order

EVENT availableLayoutsChanged(ConstStringZ layouts)

The set of currently available layouts changed

Parameters

layouts	*string*	The currently available layouts given the system state

SYNC getCurrentLayout(StringZ layout, size_t layout_size)

Returns the current layout

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 Success < 0 Normal SB failures
layout	*string*	The current layout as one of the values from getAvailableLayouts()

EVENT currentLayoutChanged(ConstStringZ layout)

The current layout changed.

Parameters

layout	*string*	The current layout as one of the values from getAvailableLayouts()

ASYNC gotoLayout(ConstStringZ layout)

Display the provided layout. Must be a value returned by getAvailableLayouts(). Posts currentLayoutChanged() if a new layout is configured.

privilege level ADMIN

Parameters

Inputs

layout	*string*	The layout to display, obtained from getAvailableLayouts().

ASYNC nextLayout()

Display the next layout. Follows the order return by getAvailableLayouts(). Posts currentLayoutChanged() if a new layout is configured.

privilege level ADMIN

ASYNC previousLayout()

Display the next layout. Follows the order return by getAvailableLayouts() Posts currentLayoutChanged() if a new layout is configured.

privilege level ADMIN

SYNC getLayoutMetadata(layouts_t *layouts)

Returns the metadata of all possible layouts, one set at a time.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	==0 Success <0 Normal SB failures.
layouts	*structure*	The metadata of all the current layouts

SYNC showPip(int show)

Sets the state of the PIP as shown or hidden

privilege level ADMIN

Parameters

Inputs

show	*integer*	1==shown 0==hidden

Outputs

_rv	*integer*	0==success, hidden 1==success, shown -EINVAL Unable to change the PIP state at this time <0 Normal switchboard failures

SYNC getPipShown(int *shown)

Gets the state of the PIP as shown or hidden

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0==success, hidden 1==success, shown <0 Normal switchboard failures
shown	*integer*	1==shown 0==hidden

EVENT pipShown(int shown)

Posted when the PIP state changes

Parameters

shown	*integer*	1==shown 0==hidden

ASYNC reloadDirectory()

Reloads all the data in the directory table.

privilege level ADMIN

ASYNC reloadFavorites()

Reloads all the data in the favorites table.

privilege level ADMIN

SYNC getOobAck()

Get the initial configutation acknowledgement state

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	return==1 the initial configuration has been acknowledged if return==0 the initial configuration has not been acknowledged if return<0 typical switchboard errors

ASYNC resetOobAck()

Resets the initial configuraiton acknownledgement back to not acknowledged

privilege level ADMIN

ASYNC ackOob()

Acknowledges the initial configuraiton

privilege level ADMIN

EVENT oobAckChanged(int value)

Posted when the initial configuration acknowledegement state changes

Parameters

value	*integer*	the value of the initial configutation acknowledgement state. see getOobAck().

SYNC getPrimarySafeAreaMargin(int* margin)

Get the safe area margin of the primary display

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0==success <=0 typical switchboard errors
margin	*integer*	The value of the horizontal margin.

SYNC setPrimarySafeAreaMargin(int margin)

Set the safe area margin of the primary display. The margin may be adjusted to the proper range and multiple

privilege level ADMIN

Parameters

Inputs

margin	*integer*	The horizontal margin to set. A value from 0-64.

Outputs

_rv	*integer*	>=0 successful, actual margin set <=0 typical switchboard errors

SYNC incrementPrimarySafeAreaMargin()

Increments the safe area margin of the primary display

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	1==success 0==already at max <=0 typical switchboard errors

SYNC decrementPrimarySafeAreaMargin()

Decrements the safe area margin of the primary display

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	1==success 0==already at min <=0 typical switchboard errors

EVENT primarySafeAreaMarginChanged(int margin)

Posted when the primary safe area margin changes

Parameters

margin	*integer*	The number of pixles in the horizontal margin

SYNC getClockFormat(StringZ buffer, size_t buffer_size)

Get the UI clock format

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM not enough room in buffer to store the clock format <0 failed to connect or other errors
buffer	*string*	The current clock format

SYNC setClockFormat(ConstStringZ value)

Set the UI clock format

privilege level ADMIN

Parameters

Inputs

value	*string*	The new clock format from: normal, or 24hour

Outputs

_rv	*integer*	0 if successful -EINVAL the value is not valid <0 failed to connect or other errors

EVENT clockFormatChanged(ConstStringZ value)

Posted when the UI clock format changes

Parameters

value	*string*	The new clock format

SYNC getLanguage(StringZ buffer, size_t buffer_size)

Get the UI language

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM not enough room in buffer to store the language <0 failed to connect or other errors
buffer	*string*	The returned language

SYNC setLanguage(ConstStringZ language)

Set the UI language.

privilege level ADMIN

Parameters

Inputs

language	*string*	The new language from: de_DE, en_US, es_ES, fr_FR, it_IT, nb_NO, pl_PL, pt_BR, ru_RU, fi_FI, sv_SE, zh_CN, zh_TW, ja_JP, or ko_KR.

Outputs

_rv	*integer*	0 if successful -EINVAL the language is not valid <0 failed to connect or other errors

EVENT languageChanged(ConstStringZ language)

Posted when the UI language changes

Parameters

language	*string*	The new language

SYNC getAdminPasscode(StringZ buffer, size_t buffer_size)

Get the admin passcode value

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM not enough room in buffer to store the passcode <0 failed to connect or other errors
buffer	*string*	The returned admin passcode

SYNC setAdminPasscode(ConstStringZ passcode)

Set the admin passcode value. The passcode must only be digits. Passcodes up to

32 digits allowed. Passcodes can start with 0. Passcodes cannot be blank.

privilege level ADMIN

Parameters

Inputs

passcode	*string*	The new passcode

Outputs

_rv	*integer*	0 if successful -EINVAL the passcode contains invalid characters

EVENT adminPasscodeChanged(ConstStringZ passcode)

Posted when the admin passcode changes

Parameters

passcode	*string*	The new passcode

SYNC getCallStartTime(time_t* value)

gets the time the current call session was started in UTC

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0==success -ENOMEM if any value parameter was NULL <0 failed to connect or other errors
value	*integer*	the start time as a time_t in UTC. if 0==value then no call is in progress

EVENT callStartTimeChanged(time_t value)

posted when the call start time changes

Parameters

value	*integer*	the start time as a time_t in UTC. if 0==value then no call is in progress

SYNC getLogLevel(int* mask)

Get the log level for the service

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if successful -ENOMEM if any output parameters were NULL <0 failed to connect or other errors
mask	*integer*	Returned mask

SYNC setLogLevel(int mask)

Set the log level for the service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	0 if successful <0 failed to connect or other errors

EVENT logLevelChanged(int mask)

Posted when the log level changes.

Parameters

mask	*integer*	returned mask

He Class Documentation

Overview

Functions by Group

Other Functions

Scheduler getLiveEvents

Scheduler

Scheduler getLiveAndSoonEvents
Scheduler getEventByUid
Scheduler getUTC
Scheduler eventLive
Scheduler eventLiveSoon
Scheduler eventExpiring
Scheduler killEvent
Scheduler eventsInserted
Scheduler eventsUpdated
Scheduler eventsDeleted

Conference and Call Methods

GuiPlayer getConferenceState
GuiPlayer conferenceStateChanged
GuiPlayer getCallDetails
GuiPlayer callDetailsChanged
GuiPlayer getCallState
GuiPlayer callStateChanged
GuiPlayer setCallInterface

Layout Management

GuiPlayer layoutGoto
GuiPlayer getCurrentLayout
GuiPlayer getRegisteredLayouts
GuiPlayer getLayouts
GuiPlayer getLayoutWindows
GuiPlayer layoutChanged
GuiPlayer getCallsInLayout
GuiPlayer callsInLayoutChanged

Verion Support

GuiPlayer isVersionCompatible 2 0
Comm terminateCall

SysAdmin

SysAdmin getActiveIPv4Config
SysAdmin ipv4ConfigUpdated

Utility

SYNC Scheduler_getLiveEvents(StringZ buffer, size_t buffer_size)

Returns a list of public conferences that are considered live. A live event is available for participants to join.

privilege level USER

Parameters

Outputs

_rv	*integer*	>=	0 if successful counting the number of live conferences
		<0	failed to connect or other errors
buffer	*string*		The newline separated list of UIDs representing the live conferences

SYNC Scheduler_getLiveAndSoonEvents(StringZ buffer, size_t buffer_size)

returns all the live and soon to be live (10 minute window) event UID's

privilege level USER

Parameters

Outputs

_rv	*integer*	>= 0 if successful counting the number of live conferences <0 failed to connect or other errors
buffer	*string*	The newline separated list of UIDs representing the conferences

SYNC Scheduler_getEventByUid(ConstStringZ uid, StringZ buffer, size_t buffer_size)

Returns the data of a particular conference identified by its UID. The conference data is structured according to the LifeSize Bridge iCalendar specification.

privilege level USER

Parameters

Inputs

uid	*string*	The uid of the conference to return.

Outputs

_rv	*integer*	>= 0 for the number of events returned (only returns 0 or 1 event) <0 failed to connect or other errors
buffer	*string*	Buffer to store a null terminated string containing a the conference in iCalendar format

SYNC Scheduler_getUTC(ConstStringZ time_str, StringZ buffer, size_t buffer_size)

Converts the ical time string into UTC

privilege level USER

Parameters

Inputs

time_str	*string*	time to convert

Outputs

_rv	*integer*	>= 0 for success <0 failed to connect or other errors
buffer	*string*	where the UTC time is returned

EVENT Scheduler_eventLive(ConstStringZ uid)

Posted when a conference becomes live.

Parameters

uid	*string*	the UID of the conference that became live

EVENT Scheduler_eventLiveSoon(ConstStringZ uid, int secondsUntilLive)

posted when an event will be live soon (within the 10 minute window) only post once. no count down needed

Parameters

uid	*string*	the UID of the conference that will become live
secondsUntilLive	*integer*	The number of seconds until the conference becomes live.

EVENT Scheduler_eventExpiring(ConstStringZ uid, int secondsRemaining)

Posted when a conference is expiring.

Parameters

uid	*string*	The UID of the conference that is expiring.

secondsRemaining	*integer*	The number of seconds remaining in the conference. 0 means the conference has expired. Once expired, no more eventExpiring events will be posted for this particular conference.

EVENT Scheduler_killEvent(ConstStringZ uid)

Conference being killed due to server issues.

Parameters

uid	*string*	The UID of the conference killed.

EVENT Scheduler_eventsInserted(ConstStringZ uids, ConstStringZ transId)

Posted when a set of new conferences have been inserted into the calendar. Calling Scheduler_saveEventLong() may trigger this event.

Parameters

uids	*string*	A newline separated list of UIDs that map to the conferences that were inserted.
transId	*string*	The transId of the matching call to Scheduler_saveEventLong().

EVENT Scheduler_eventsUpdated(ConstStringZ uids, ConstStringZ transId)

Posted when a set of conferences have been updated in the calendar. Calling Scheduler_saveEventLong() may trigger this event.

Parameters

uids	*string*	A newline separated list of UIDs that map to the conferences that were updated.
transId	*string*	The transId of the matching call to Scheduler_saveEventLong().

EVENT Scheduler_eventsDeleted(ConstStringZ uids)

Posted when a set of conferences have been deleted from the calendar. Calling Scheduler_removeEvents() may trigger this event.

Parameters

uids	*string*	A newline separated list of UIDs that map to the conferences that were updated.

SYNC GuiPlayer_getConferenceState(ConstStringZ confUid, lsconfstate_t* state)

Returns the state of a conference.

privilege level USER

Parameters

Inputs

confUid	*string*	The UID of the conference to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found <0 failed to connect or other errors
state	*structure*	The state of the conference. See lsconfstate_t for details.

EVENT GuiPlayer_conferenceStateChanged(const lsconfstate_t* state)

Posted when the state of a conference changes.

Parameters

state	*structure*	The state of the conference. See lsconfstate_t for details.

SYNC GuiPlayer_getCallDetails(int callId, lscalldetails_t* details)

Returns the details of a call.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found <0 failed to connect or other errors
details	*structure*	The details of the call. See lscalldetails_t for details.

EVENT GuiPlayer_callDetailsChanged(const lscalldetails_t* details)

Posted when the details of a call change.

Parameters

details	*structure*	The details of the call. See lscalldetails_t for details.

SYNC GuiPlayer_getCallState(int callId, lscallstate_t* state)

Returns the state of a call.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found <0 failed to connect or other errors
state	*structure*	The state of the call. See lscallstate_t for details.

EVENT GuiPlayer_callStateChanged(const lscallstate_t* state)

Posted when the state of a call changes.

Parameters

state	*structure*	The state of the call. See lscallstate_t for details.

SYNC GuiPlayer_setCallInterface(int callId, ConstStringZ language)

Configures an endpoint in control over the call's user interface by doing the following:
- Sets the call's language.
- Disables all helium UI graphics for the call.
- Disables all user interaction via DTMF or FECC, defaults to DTMF.
- Disables self view regardless of conference setting.
- Prevents the VOP from answering(currently does nothing)

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to interface with
language	*string*	Use one of these values:de_DE, en_US, es_ES, fr_FR, it_IT, no_NO, pl_PL, pt_BR, ru_RU, fi_FI, sv_SV, zh_CN_SM, zh_CN_TR, ja_JA, ko_KO.

Outputs

_rv	*integer*	0 if successful

-ENOENT if callId is not found

SYNC GuiPlayer_layoutGoto(int callId, ConstStringZ name)

Display a specific layout for the call. Use GuiPlayer_getLayouts(), GuiPlayer_getLayoutsConference(), or GuiPlayer_getScenarioLayouts() to get a list of layout names to use.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the call ID of the call to affect
name	*string*	the name of the conference to display

Outputs

_rv	*integer*	>=0 if the successful -ENOENT if the call is not found

SYNC GuiPlayer_getCurrentLayout(int callId, StringZ buffer, size_t buffer_size)

Returns the name of the current layout for the call. The names can be used in the GuiPlayer_layoutGoto() method.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the call ID of the call to query

Outputs

_rv	*integer*	the number of layout names included in buffer -ENOENT if the call is not found
buffer	*string*	the return buffer with the newline separated list of layouts

SYNC GuiPlayer_getRegisteredLayouts(StringZ buffer, size_t buffer_size)

Returns a newline separated list of all available layout names. The names can be used in the GuiPlayer_layoutGoto() method. NOTE: not all layouts may be appropriate for a given conference or call.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	the number of layout names included in buffer
buffer	*string*	the return buffer with the newline separated list of

SYNC GuiPlayer_getLayouts(int callId, StringZ buffer, size_t buffer_size)

Returns a newline separated list of layout names appropriate for the call. The names can be used in the GuiPlayer_layoutGoto() method.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the call ID of the call to query

Outputs

_rv	*integer*	the number of layout names included in buffer -ENOENT if the call is not found
buffer	*string*	the return buffer with the newline separated list of layouts

SYNC GuiPlayer_getLayoutWindows(ConstStringZ layout_name, lsvidwindows_t* windows)

Returns window information about the current layout of a call.

privilege level ADMIN

Parameters

Inputs

layout_name	*string*	The name of the layout to query.

Outputs

_rv	*integer*	Return status (0 = success)
windows	*structure*	See lsvidwindows_t for details of the data returned.

EVENT GuiPlayer_layoutChanged(int callId, ConstStringZ name)

Posted when a call changes to a different video layout screen. It is not posted due to talker reording on the same screen.

Parameters

callId	*integer*	The ID of the call affected.
name	*string*	The name of the new layout

ASYNC GuiPlayer_getCallsInLayout(int callId)

Causes the callInLayoutChanged event to be emitted.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The ID of the call to get the layout info from.

EVENT GuiPlayer_callsInLayoutChanged(int callId, ConstStringZ calls)

Specifies how a particular call's layout is currently populated.

Parameters

callId	*integer*	**UNDOCUMENTED**
calls	*string*	An Comma deliminated string of callIds to populate the current layout with.

SYNC GuiPlayer_isVersionCompatible_2_0()

determines if the current runtime is compatible with Version 2.0.x. use the WebServices.hasMethod() to test for compatibility

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if not compatible, 1 if compatible

ASYNC Comm_terminateCall(unsigned int commCallId, int reasonCode)

Interface to terminate ongoing

privilege level ADMIN

Parameters

Inputs

commCallId	*unsigned int*	Comm callId of the call to be transferred
reasonCode	*integer*	The reason for call termination

SYNC SysAdmin_getActiveIPv4Config(int *ifnum, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Retrieve the active IP configuration for Helium. This is bond0 if bonding is enabled, otherwise, it is the first valid ip address.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
ifnum	*integer*	of Active interface.
address	*string*	IP address of interface
netmask	*string*	netmask of interface
broadcast	*string*	broadcast address of interface

EVENT SysAdmin_ipv4ConfigUpdated()

Notification that the active configuration of a network interface has changed

SYNC SB_listen(ConstStringZ event)

Listen for a particular event

privilege level ADMIN

Parameters

Inputs

event	*string*	The event to listen to

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getProxyStatus(int *status)

Get the current proxy status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)	
status	*integer*	0	proxy is down
		1	proxy is up

EVENT proxyStatusChanged(int proxyUp)

issued when the proxy goes up or down.

Parameters

proxyUp	*integer*	0	proxy is down
		1	proxy is up

He2 Class Documentation

Overview

Functions by Group

Other Functions

Scheduler getLiveEvents

Scheduler

Scheduler getLiveAndSoonEvents
Scheduler getEventByUid
Scheduler getUTC
Scheduler eventLive
Scheduler eventLiveSoon
Scheduler eventExpiring
Scheduler killEvent
Scheduler eventsInserted
Scheduler eventsUpdated
Scheduler eventsDeleted

Conference and Call Methods

GuiPlayer getConferenceState
GuiPlayer conferenceStateChanged
GuiPlayer getCallDetails
GuiPlayer callDetailsChanged
GuiPlayer getCallState
GuiPlayer callStateChanged
GuiPlayer setCallInterface

Layout Management

GuiPlayer layoutGoto
GuiPlayer getCurrentLayout
GuiPlayer getRegisteredLayouts
GuiPlayer getLayouts
GuiPlayer getLayoutWindows
GuiPlayer layoutChanged
GuiPlayer getCallsInLayout
GuiPlayer callsInLayoutChanged

Verion Support

GuiPlayer isVersionCompatible 2 0
Comm terminateCall

SysAdmin

SysAdmin getActiveIPv4Config
SysAdmin ipv4ConfigUpdated

Utility

SYNC Scheduler_getLiveEvents(StringZ buffer, size_t buffer_size)

Returns a list of public conferences that are considered live. A live event is available for participants to join.

privilege level USER

Parameters

Outputs

_rv	*integer*	>=	0 if successful counting the number of live conferences
		<0	failed to connect or other errors
buffer	*string*		The newline separated list of UIDs representing the live conferences

SYNC Scheduler_getLiveAndSoonEvents(StringZ buffer, size_t buffer_size)

returns all the live and soon to be live (10 minute window) event UID's

privilege level USER

Parameters

Outputs

_rv	*integer*	>= 0 if successful counting the number of live conferences <0 failed to connect or other errors
buffer	*string*	The newline separated list of UIDs representing the conferences

SYNC Scheduler_getEventByUid(ConstStringZ uid, StringZ buffer, size_t buffer_size)

Returns the data of a particular conference identified by its UID. The conference data is structured according to the LifeSize Bridge iCalendar specification.

privilege level USER

Parameters

Inputs

uid	*string*	The uid of the conference to return.

Outputs

_rv	*integer*	>= 0 for the number of events returned (only returns 0 or 1 event) <0 failed to connect or other errors
buffer	*string*	Buffer to store a null terminated string containing a the conference in iCalendar format

SYNC Scheduler_getUTC(ConstStringZ time_str, StringZ buffer, size_t buffer_size)

Converts the ical time string into UTC

privilege level USER

Parameters

Inputs

time_str	*string*	time to convert

Outputs

_rv	*integer*	>= 0 for success <0 failed to connect or other errors
buffer	*string*	where the UTC time is returned

EVENT Scheduler_eventLive(ConstStringZ uid)

Posted when a conference becomes live.

Parameters

uid	*string*	the UID of the conference that became live

EVENT Scheduler_eventLiveSoon(ConstStringZ uid, int secondsUntilLive)

posted when an event will be live soon (within the 10 minute window) only post once. no count down needed

Parameters

uid	*string*	the UID of the conference that will become live
secondsUntilLive	*integer*	The number of seconds until the conference becomes live.

EVENT Scheduler_eventExpiring(ConstStringZ uid, int secondsRemaining)

Posted when a conference is expiring.

Parameters

uid	*string*	The UID of the conference that is expiring.

secondsRemaining	*integer*	The number of seconds remaining in the conference. 0 means the conference has expired. Once expired, no more eventExpiring events will be posted for this particular conference.

EVENT Scheduler_killEvent(ConstStringZ uid)

Conference being killed due to server issues.

Parameters

uid	*string*	The UID of the conference killed.

EVENT Scheduler_eventsInserted(ConstStringZ uids, ConstStringZ transId)

Posted when a set of new conferences have been inserted into the calendar. Calling Scheduler_saveEventLong() may trigger this event.

Parameters

uids	*string*	A newline separated list of UIDs that map to the conferences that were inserted.
transId	*string*	The transId of the matching call to Scheduler_saveEventLong().

EVENT Scheduler_eventsUpdated(ConstStringZ uids, ConstStringZ transId)

Posted when a set of conferences have been updated in the calendar. Calling Scheduler_saveEventLong() may trigger this event.

Parameters

uids	*string*	A newline separated list of UIDs that map to the conferences that were updated.
transId	*string*	The transId of the matching call to Scheduler_saveEventLong().

EVENT Scheduler_eventsDeleted(ConstStringZ uids)

Posted when a set of conferences have been deleted from the calendar. Calling Scheduler_removeEvents() may trigger this event.

Parameters

uids	*string*	A newline separated list of UIDs that map to the conferences that were updated.

SYNC GuiPlayer_getConferenceState(ConstStringZ confUid, lsconfstate_t* state)

Returns the state of a conference.

privilege level USER

Parameters

Inputs

confUid	*string*	The UID of the conference to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found <0 failed to connect or other errors
state	*structure*	The state of the conference. See lsconfstate_t for details.

EVENT GuiPlayer_conferenceStateChanged(const lsconfstate_t* state)

Posted when the state of a conference changes.

Parameters

state	*structure*	The state of the conference. See lsconfstate_t for details.

SYNC GuiPlayer_getCallDetails(int callId, lscalldetails_t* details)

Returns the details of a call.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found <0 failed to connect or other errors
details	*structure*	The details of the call. See lscalldetails_t for details.

EVENT GuiPlayer_callDetailsChanged(const lscalldetails_t* details)

Posted when the details of a call change.

Parameters

details	*structure*	The details of the call. See lscalldetails_t for details.

SYNC GuiPlayer_getCallState(int callId, lscallstate_t* state)

Returns the state of a call.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to query.

Outputs

_rv	*integer*	>=0 if successful -ENOENT if conference is not found <0 failed to connect or other errors
state	*structure*	The state of the call. See lscallstate_t for details.

EVENT GuiPlayer_callStateChanged(const lscallstate_t* state)

Posted when the state of a call changes.

Parameters

state	*structure*	The state of the call. See lscallstate_t for details.

SYNC GuiPlayer_setCallInterface(int callId, ConstStringZ language)

Configures an endpoint in control over the call's user interface by doing the following:
- Sets the call's language.
- Disables all helium UI graphics for the call.
- Disables all user interaction via DTMF or FECC, defaults to DTMF.
- Disables self view regardless of conference setting.
- Prevents the VOP from answering(currently does nothing)

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The call ID of the call to interface with
language	*string*	Use one of these values:de_DE, en_US, es_ES, fr_FR, it_IT, no_NO, pl_PL, pt_BR, ru_RU, fi_FI, sv_SV, zh_CN_SM, zh_CN_TR, ja_JA, ko_KO.

Outputs

_rv	*integer*	0 if successful

-ENOENT if callId is not found

SYNC GuiPlayer_layoutGoto(int callId, ConstStringZ name)

Display a specific layout for the call. Use GuiPlayer_getLayouts(), GuiPlayer_getLayoutsConference(), or GuiPlayer_getScenarioLayouts() to get a list of layout names to use.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the call ID of the call to affect
name	*string*	the name of the conference to display

Outputs

_rv	*integer*	>=0 if the successful -ENOENT if the call is not found

SYNC GuiPlayer_getCurrentLayout(int callId, StringZ buffer, size_t buffer_size)

Returns the name of the current layout for the call. The names can be used in the GuiPlayer_layoutGoto() method.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the call ID of the call to query

Outputs

_rv	*integer*	the number of layout names included in buffer -ENOENT if the call is not found
buffer	*string*	the return buffer with the newline separated list of layouts

SYNC GuiPlayer_getRegisteredLayouts(StringZ buffer, size_t buffer_size)

Returns a newline separated list of all available layout names. The names can be used in the GuiPlayer_layoutGoto() method. NOTE: not all layouts may be appropriate for a given conference or call.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	the number of layout names included in buffer
buffer	*string*	the return buffer with the newline separated list of

SYNC GuiPlayer_getLayouts(int callId, StringZ buffer, size_t buffer_size)

Returns a newline separated list of layout names appropriate for the call. The names can be used in the GuiPlayer_layoutGoto() method.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	the call ID of the call to query

Outputs

_rv	*integer*	the number of layout names included in buffer -ENOENT if the call is not found
buffer	*string*	the return buffer with the newline separated list of layouts

SYNC GuiPlayer_getLayoutWindows(ConstStringZ layout_name, lsvidwindows_t* windows)

Returns window information about the current layout of a call.

privilege level ADMIN

Parameters

Inputs

layout_name	*string*	The name of the layout to query.

Outputs

_rv	*integer*	Return status (0 = success)
windows	*structure*	See lsvidwindows_t for details of the data returned.

EVENT GuiPlayer_layoutChanged(int callId, ConstStringZ name)

Posted when a call changes to a different video layout screen. It is not posted due to talker reording on the same screen.

Parameters

callId	*integer*	The ID of the call affected.
name	*string*	The name of the new layout

ASYNC GuiPlayer_getCallsInLayout(int callId)

Causes the callInLayoutChanged event to be emitted.

privilege level ADMIN

Parameters

Inputs

callId	*integer*	The ID of the call to get the layout info from.

EVENT GuiPlayer_callsInLayoutChanged(int callId, ConstStringZ calls)

Specifies how a particular call's layout is currently populated.

Parameters

callId	*integer*	**UNDOCUMENTED**
calls	*string*	An Comma deliminated string of callIds to populate the current layout with.

SYNC GuiPlayer_isVersionCompatible_2_0()

determines if the current runtime is compatible with Version 2.0.x. use the WebServices.hasMethod() to test for compatibility

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 if not compatible, 1 if compatible

ASYNC Comm_terminateCall(unsigned int commCallId, int reasonCode)

Interface to terminate ongoing

privilege level ADMIN

Parameters

Inputs

commCallId	*unsigned int*	Comm callId of the call to be transferred
reasonCode	*integer*	The reason for call termination

SYNC SysAdmin_getActiveIPv4Config(int *ifnum, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Retrieve the active IP configuration for Helium. This is bond0 if bonding is enabled, otherwise, it is the first valid ip address.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
ifnum	*integer*	of Active interface.
address	*string*	IP address of interface
netmask	*string*	netmask of interface
broadcast	*string*	broadcast address of interface

EVENT SysAdmin_ipv4ConfigUpdated()

Notification that the active configuration of a network interface has changed

SYNC SB_listen(ConstStringZ event)

Listen for a particular event

privilege level ADMIN

Parameters

Inputs

event	*string*	The event to listen to

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getProxyStatus(int *status)

Get the current proxy status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)	
status	*integer*	0	proxy is down
		1	proxy is up

EVENT proxyStatusChanged(int proxyUp)

issued when the proxy goes up or down.

Parameters

proxyUp	*integer*	0	proxy is down
		1	proxy is up

IR Class Documentation

Overview

The IR daemon translates IR keypresses into switchboard events.

Functions by Group

IR Events

Setting the Log Level

EVENT keyDown(int key, int remote_id)

Issued when an IR keypress is received.

Parameters

key	*integer*	Key code
remote_id	*integer*	Remote control identifier

EVENT keyRepeat(int key, int remote_id)

Issued when an IR key is repeated on the remote.

Parameters

key	*integer*	Key code
remote_id	*integer*	Remote control identifier

EVENT keyUp(int key, int remote_id)

Issued when an IR key is released on the remote.

Parameters

key	*integer*	Key code
remote_id	*integer*	Remote control identifier

SYNC getIREnabled()

Determine if IR is enabled.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Result status

- 0 Success, current state is disabled
- 1 Success, current state is enabled

SYNC setIREnabled(int enabled)

Enable or disable IR. See http://www.sqlite.org/c3ref/exec.html for info on "other"

privilege level ADMIN

Parameters

Inputs

enabled	*integer*	New state (0 == false, 1 == true)

Outputs

_rv	*integer*	Result status

- 0 Success, current state is disabled
- 1 Success, current state is enabled
- -1 Failure, input parameter invalid, IR state not changed
- other Failure, IR state not changed

EVENT IREnabledChanged()

Notification event issued when IR enablement state is changed.

SYNC setIRFiltered(int who)

Filter the IR events by remote type.

privilege level ADMIN

Parameters

Inputs

who	*integer*	Which remote to allow to send IR to the system.

any=0	any remote
apple=1	Apple remote
silver=2	LS Silver
black=3	LS Black
rib=4	New LS Remote (Rib)

Outputs

_rv	*integer*	Result status

0	Success, IR is now filtered as appropriate
- 1	Failure, invalid IR device selected, no changes applied

SYNC getIRFiltered()

Determine the current IR remote filter.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Current filter or negative error

any=0	any remote
apple=1	Apple remote
silver=2	LS Silver
black=3	LS Black
rib=4	New LS Remote (Rib)

EVENT filterChanged()

Issued when the IR remote filter is changed.

SYNC setLogLevel(int mask)

Set the log level for the IR manager service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

returns the current log level mask of the IT manager

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
mask	*integer*	returned mask

EVENT logLevelChanged(int mask)

issued when the IR manager log level changes

Parameters

mask	*integer*	new log level mask

SYNC getLineInStatus()

Gets the current line-in status.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Line-in status value 0 line-in is not active value 1 line-in is active

EVENT lineInStatusChanged(int status)

Issued when status of line-in changes.

Parameters

status	*integer*	Line-in status value 0 line-in is not active value 1 line-in is active

LDAP_Directory Class Documentation

Overview

A directory service actor enables access to a database of directory entries. This can be a local database or a remote database. Each database must have a unique name which is used to register the database with the master directory service. This requires that the initialization of the directory server object use the name in the C++ constructor. Example:

```
Directory_server *srv = new Directory_server("skype_directory")
```

Every database is expected to maintain a locally unique id for each entry in the database. This id is always represented as a string within the context of the directory service but may be an integer or some other type in the actual database. Contact info is represented as a JSON structure with the following fields: "id", "firstname", "lastname", "system", "refid", "number", "type", "protocol", and "bandwidth". These fields may be mapped by the directory service to other fields in the actual remote database, or in some cases, synthesized from fields in the remote database (i.e. firstname and lastname could be derived from a single name field, or dial could be a combination of several fields).

Directory services should use the client methods described in "Managing Directory Services" to register with the master server. Only the master should implement the server methods described there.

Functions by Group

Querying the directory database

beginQuery
itemsRemaining
abortQuery
getNext
requestNext
queryResult

Editing the database

saveEntry
saveEntryDone
deleteEntry
deleteEntryDone
clearAllEntries
clearAllEntriesDone
setConfiguration
getConfiguration
configurationChanged
setLogLevel
getLogLevel
logLevelChanged

presenceChanged
entryCreated
entryUpdated
entryDeleted
reload

Managing directory services

registerServer
deleteServer
getServerEnumerator
enumerateNextServer
requestServers

SYNC beginQuery(ConstStringZ query)

Start a directory lookup operation. The query is a pattern string. The query should return all records where the pattern matches the first characters in any word of the fields "firstname", "lastname", or "systemname". Words are defined as contiguous sequences of alphanumeric characters, so a systemname of "CR-AUS-Alamo" should match the query "ala". Queries are case insensitive.

privilege level ADMIN

Parameters

Inputs

query	*string*	Query specification

Outputs

_rv	*integer*	Query handle or negative error code

SYNC itemsRemaining(int handle)

Get the number of result rows remaining in an active query. This may be an approximation, due to additional filtering needed on the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Number of result entries, or negative error code

ASYNC abortQuery(int handle)

Close out a query before fetching all of the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

SYNC getNext(int handle, StringZ buffer, size_t buffer_size)

Retrieve the next result group from a directory query synchronously. Interchangable with requestNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Result status
		0 Success
		-1 End of query
		-EAGAIN Query not ready, try again
		-EINVAL Bad query handle
		-E2BIG Buffer too small
buffer	*string*	Buffer to accept query results. Buffer will be formatted with a JSON array of structures representing entries. Each entry will have the fields requested in the query.

ASYNC requestNext(int handle)

Request the next result group from a directory query asynchronously. Interchangable with getNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

RESPONSE queryResult(int handle, int status, ConstStringZ result)

Response to requestNext containing query results.

Parameters

handle	*integer*	Query handle used in requestNext
status	*integer*	Query status
		0 Success
		-1 End of query

-EINVAL Bad query handle

result *string* A JSON array of structures representing directory entries matching the query.

ASYNC saveEntry(ConstStringZ info)

Create a new entry in the database or update an existing entry. This will normally only be possible for the local directory. If the entry contains an id field that matches an existing entry in the database then this will perform an update, otherwise it will create the entry.

privilege level ADMIN

Parameters

Inputs

info *string* Contact info record in JSON format

RESPONSE saveEntryDone(int result, ConstStringZ info)

Signals the end of a saveEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded
- -EPERM Operation not allowed
- -EINVAL Invalid info record

info *string* JSON data for contact with directory added id field if no id was present in the original entry.

ASYNC deleteEntry(ConstStringZ id)

Delete an entry in the database. This will normally only be possible for the local directory.

privilege level ADMIN

Parameters

Inputs

id *string* Id field of record to delete.

RESPONSE deleteEntryDone(int result)

Signals the end of a deleteEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded

-EPERM	Operation not allowed
-EINVAL	Invalid info record

ASYNC clearAllEntries()

Remove all entries from the directory.

privilege level ADMIN

RESPONSE clearAllEntriesDone()

Signals the end of a clear-all operation.

SYNC setConfiguration(ConstStringZ config)

Configure remote server parameters. The format of the configuration object can be different for each type of directory service. In general it may include a server address, username, password, and base search string. The local directory does not require this command.

privilege level ADMIN

Parameters

Inputs

config	*string*	JSON formatted configuration object

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getConfiguration(StringZ config, int config_size)

Retrieve the current configuration.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error
config	*string*	Buffer to receive current configuration

EVENT configurationChanged(ConstStringZ server, ConstStringZ config)

Issued when the configuration is changed by setConfiguration.

Parameters

server

	string	Name of reconfigured server
config	*string*	New configuration

SYNC setLogLevel(int mask)

Set the logging mask.

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New logging mask

Outputs

_rv	*integer*	0 or negative error code

SYNC getLogLevel(int *mask)

Retrieve the current logging mask

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error code.
mask	*integer*	logging mask

EVENT logLevelChanged(int mask)

Issued when the logging level is changed.

Parameters

mask	*integer*	New logging level

EVENT presenceChanged(ConstStringZ server, ConstStringZ entries)

Issued to indicate a change of presence for one or more entries.

Parameters

server	*string*	Directory service that provides this entry
entries	*string*	JSON array of entries changed

EVENT entryCreated(ConstStringZ server, ConstStringZ info)

issued when a new entry is created in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Created entry

EVENT entryUpdated(ConstStringZ server, ConstStringZ info)

issued when an entry is updated in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Updated entry

EVENT entryDeleted(ConstStringZ server, ConstStringZ info)

issued when an entry is deleted from the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Deleted entry

EVENT reload(ConstStringZ server)

issued when cached entries for this directory should be invalidated.

Parameters

server	*string*	Actor name for this directory LISTEN Directory_requestServers()

SYNC registerServer(ConstStringZ serverName)

Add a directory server to the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of new server

Outputs

_rv	*integer*	Result status

0	Success
negative	Negative error code

SYNC deleteServer(ConstStringZ serverName)

Remove a directory server from the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of server to remove.

Outputs

_rv	*integer*	Result status
	0	Success
	-ENOENT	Server name not found

SYNC getServerEnumerator()

Begin enumeration of directory servers.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextServer(int handle, StringZ buffer, size_t buffer_size)

Get the next set of servers in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getServerEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has one or more server names.
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive server names. Names are space seperated. As many names as will fit in the buffer are returned.

EVENT requestServers()

Sent by the master directory to request that all directory servers register their name.

Led Class Documentation

Overview

Functions by Group

Other Functions

SYNC setLedPattern(LedPattern pattern)

Set led output pattern

privilege level USER

Parameters

Inputs

pattern *unsigned int*

Outputs **_rv** *integer*

SYNC getLedPattern(LedPattern* pattern)

Get led output pattern

privilege level USER

Parameters

Outputs

_rv *integer*

pattern *unsigned int*

EVENT ledPatternChanged(LedPattern pattern)

Event that indicates the LED pattern has changed

Parameters

pattern *unsigned int*

SYNC setBacklightBrightness(int brightness)

Set backlight brightness

privilege level USER

Parameters

Inputs

brightness	*integer*	Backlight brightness percentage 0 is off and 100 is full on

Outputs

_rv	*integer*

SYNC getBacklightBrightness(int* brightness)

Get backlight brightness

privilege level USER

Parameters

Outputs

_rv	*integer*

brightness *integer*Backlight brightness percentage 0 is off and 100 is full on

EVENT backlightBrightnessChanged(int brightness)

Event sent after backlight brightness changes

Parameters

brightness	*integer*	Backlight brightness percentage 0 is off and 100 is full on

License Class Documentation

Overview

The license manager maintains system licenses.

Functions by Group

Managing licenses

Setting the Log Level

ASYNC addLicense(ConstStringZ license)

Verifies supplied license text and installs as a license if it verifies. If another license of the same type was already installed, it will be replaced by the new license. Expect License_addDone response and License_infoUpdated event.

privilege level ADMIN

Parameters

Inputs

license	*string*	License string, e.g., as copied from license server response to query such as http://software.lifesize.com/license/lsmaint.php?sn=

RESPONSE addDone(int result, ConstStringZ type)

Response to License_add request.

Parameters

`result` *integer* Result of addLicense operation

0	Success
-1	Fail, OpenSSL invalid digest type
-2	Fail, Unable to allocate OpenSSL BIO
-3	Fail, Unable to open public key file
-4	Fail, Unable to read public key file
-5	Fail, Verification failed
-6	Fail, OpenSSL verify update failed
-7	Fail, OpenSSL verify init failed
-12	Fail, Invalid license format
-13	Fail, Unable to allocate signature buffer
-14	Fail, License param NULL or empty
-17	Fail, System has no serial number
-21	Fail, Serial number invalid
-22	Fail, Busy try again later
-23	Fail, No thread available
-24	Fail, No memory available
-25	Fail, License already exists
-26	Fail, License type is not applicable to target system
-27	Fail, License type is not supported
-30	Fail, Adding license would exceed port capacity maximum
-31	Fail, Adding license requires port capacity minimum of 16

`type` *string* The type of license

"maint"	Maintenance type license
"feature_expand"	Expanded conference feature license
"feature_clustering"	Clustering feature license
"capacity_port_plus4-1"	Port capacity 4-port

	bump license
"capacity_port_plus4-2"	Port capacity 4-port bump license
"feature_dual_display"	Dual display license
"feature_resolution_1080p"	1080p resolution license

ASYNC updateLicense(void)

Cause all appropriate licenses (based on system serial number) to be downloaded from the license server and installed on the system. Any already-installed license of the same type as a downloaded license will be replaced. Expect License_updateDone response and License_infoUpdated events.

privilege level ADMIN

RESPONSE updateDone(int result)

Response to License_update request.

Parameters

result *integer* Result status of updateLicense

0	Success, all licenses updated
-1	Fail, OpenSSL invalid digest type
-2	Fail, Unable to allocate OpenSSL BIO
-3	Fail, Unable to open public key file
-4	Fail, Unable to read public key file
-5	Fail, Verification failed
-6	Fail, OpenSSL verify update failed
-7	Fail, OpenSSL verify init failed
-12	Fail, Invalid license format
-13	Fail, Unable to allocate signature buffer
-14	Fail, License param NULL or empty
-17	Fail, System has no serial number
-21	Fail, Serial number invalid
-22	Fail, Busy try again later
-23	Fail, No thread available
-24	Fail, No memory available

ASYNC deleteLicense(ConstStringZ type)

Delete the currently-installed license of the specified type, if any. Expect License_deleteDone response and License_infoUpdated event.

privilege level ADMIN

Parameters

Inputs

`type` *string*

`RESPONSE deleteDone( int result, ConstStringZ type )`

Response to License_delete request.

Parameters

`result` *integer* Result of deleteLicense operation

0	Success
-15	Fail, License type param NULL, empty or does not exist
-19	Fail, Internal data store error indicated
-22	Fail, Busy try again later
-23	Fail, No thread available
-24	Fail, No memory available
-28	Fail, License does not exist
-29	Fail, License removal restricted by port capacity minimum of 16

`type` *string* The type of license

"maint"	Maintenance type license
"feature_expand"	Expanded conference feature license
"feature_clustering"	Clustering feature license
"capacity_port_plus4-1"	Port capacity 4-port bump license
"capacity_port_plus4-2"	Port capacity 4-port bump license
"feature_dual_display"	Dual display license
"feature_resolution_1080p"	1080p resolution license

`ASYNC validate(ConstStringZ type, ConstStringZ date)`

Request validation of the specified license type on date specified. Expect

License_validateDone response.

privilege level ADMIN

Parameters

Inputs

`type`	*string*	The type of license

"maint"	Maintenance type license
"feature_expand"	Expanded conference feature license
"feature_clustering"	Clustering feature license
"capacity_port_plus4-1"	Port capacity 4-port bump license
"capacity_port_plus4-2"	Port capacity 4-port bump license
"feature_dual_display"	Dual display license
"feature_resolution_1080p"	1080p resolution license

`date`	*string*	Date license is to be validated for

`RESPONSE validateDone( int result, ConstLicenseInfo info )`

Response to License_validate request.

Parameters

`result`	*integer*	Result of validate operation

0	Success, info contains valid data
-1	Fail, OpenSSL invalid digest type
-2	Fail, Unable to allocate OpenSSL BIO
-3	Fail, Unable to open public key file
-4	Fail, Unable to read public key file
-5	Fail, Verification failed
-6	Fail, OpenSSL verify update failed
-7	Fail, OpenSSL verify init failed
-8	Fail, Query date format incorrect - should be 'mm/dd/yyyy'
-9	Fail, License date format incorrect - should be 'yyyy-mm-dd'
-10	Fail, Unable to find license date
-11	Fail, Validation failed
-12	Fail, Invalid license format

- -13 Fail, Unable to allocate signature buffer
- -15 Fail, License type param NULL or empty
- -17 Fail, Serial number doesnt exist
- -20 Fail, Date param is NULL or empty
- -21 Fail, Serial number invalid
- -22 Fail, Busy try again later
- -23 Fail, No thread available
- -24 Fail, No memory available

info *structure* Structure holding info about matched license, only valid if result is 0

license[1024]	The license text and md5 signed digest signature
version[64]	Version license applies to
type[32]	License type "maint", "feature_expand", "feature_clustering", "capacity_port_plus4-1", "capacity_port_plus4-2", "feature_dual_display", "feature_resolution_1080p"
serialNumber[32]	Serial number license applies to
utc_expires	Time_t UTC last valid day, month, year for license
utc_date2	UTC date unknown use
utc_date3	UTC date unknown use
utc_date4	UTC date unknown use
signature[1024]	MD5 signed digest signature
remote	license location 0 license is on local device 1 license is on remote server

SYNC getLicenseEnumerator(ConstStringZ type)

returns a handle (positive integer) to use for enumeration of licenses

privilege level ADMIN

Parameters

Inputs

type	*string*	The type of license	
		""	Empty string means enumerate all license types
		"maint"	Maintenance type license
		"feature_expand"	Expanded conference feature license
		"feature_clustering"	Clustering feature license
		"capacity_port_plus4-1"	Port capacity 4-port bump license
		"capacity_port_plus4-2"	Port capacity 4-port bump license
		"feature_dual_display"	Dual display license
		"feature_resolution_1080p"	1080p resolution license

Outputs

_rv	*integer*	handle to use for enumeration

`SYNC enumerateNextLicense(int handle, LicenseInfo *info)`

retrieve the next license in an enumeration

privilege level ADMIN

Parameters

Inputs

handle	*integer*	the handle returned by getLicenseEnumerator

Outputs

_rv	*integer*	Return status	
		0	Success
		-1	End of enumeration or error
info	*structure*	Structure holding info about matched license	
		license[1024]	The license text and md5 signed digest signature
		version[64]	Version license applies to
		type[32]	License type "maint", "feature_expand",

"feature_clustering", "capacity_port_plus4-1", "capacity_port_plus4-2", "feature_dual_display", "feature_resolution_1080p"

serialNumber[32]	Serial number license applies to
utc_expires	Time_t UTC last valid day, month, year for license
utc_date2	UTC date unknown use
utc_date3	UTC date unknown use
utc_date4	UTC date unknown use
signature[1024]	MD5 signed digest signature
remote	license location 0 license is on local device 1 license is on remote server

`SYNC setLogLevel(int mask)`

Set the log level for the pref service

privilege level ADMIN

Parameters

Inputs

`mask`	*integer*	New log mask

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC getLogLevel(int *mask)`

returns the current log level mask of dataman

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`mask`	*integer*	returned mask

`EVENT logLevelChanged(int mask)`

issued when the data manager log level changes

Parameters

mask	*integer*	new log level mask

EVENT infoUpdated(ConstStringZ type)

Occurs after a license is added, updated, or deleted.

Parameters

type	*string*

SYNC getDualDisplayLicensed()

determine if the system is licensed for dual display capability or not

privilege level USER

Parameters

Outputs

_rv	*integer*	License status 0 System is not licensed for dual display 1 System is licensed for dual display

SYNC getResolution1080pLicensed()

determine if the system is licensed for 1080p resolution or not

privilege level USER

Parameters

Outputs

_rv	*integer*	License status 0 System is not licensed for 1080p resolution 1 System is licensed for 1080p resolution

Lifelink Class Documentation

Overview

Functions by Group

Other Functions

SYNC getDevs(struct ll_dev *devs, int *devs_count)

Returns a list of all connected Lifelink devices

privilege level ADMIN

Parameters

Inputs

devs_count	*unsigned int*	The number of device structs returned

Outputs

_rv	*integer*	Return status (0 = success)
devs	*array of structure*	A list of device structs, should be LL_MAX_DEVS long

SYNC getAttrDef(enum ll_attrs attrId, struct ll_attr *attrOut)

Get the definition for the specified attr

privilege level ADMIN

Parameters

Inputs

attrId	*unsigned int*	Number identifying an attr to get the definition of

Outputs

_rv	*integer*	Return status (0 = success)
attrOut	*structure*	Filled in with the attr's definition

SYNC getAttrByName(ConstStringZ attrName, struct ll_attr *attrOut)

Get the definition of the attr specified by name

privilege level ADMIN

Parameters

Inputs

attrName	*string*	Name of the attribute

Outputs

_rv	*integer*	Return status (0 = success)
attrOut	*structure*	Filled in with the attr's definition

SYNC getAttrList(ConstStringZ dev, struct ll_attr *attrList, int *attrList_count)

Retrieve a list of attributes supported by the given device

privilege level ADMIN

Parameters

Inputs

dev	*string*	The device whose attribute list you want
attrList_count	*unsigned int*	The number of attributes returned

Outputs

_rv	*integer*	Return status (0 = success)
attrList	*array of structure*	Array in which to fill in the supported attribute metadata

SYNC getDevAttr(const struct ll_attr *attr, StringZ val, size_t val_size)

Get a device's attribute specified in the struct

privilege level ADMIN

Parameters

Inputs

attr	*structure*	Struct identifying the dev and attr you want

Outputs

_rv	*integer*	Return status (0 = success)
val	*string*	The value of the specified attribute

SYNC setDevAttr(const struct ll_attr *attr, ConstStringZ val)

Set a device's attribute

privilege level ADMIN

Parameters

Inputs

attr	*structure*	identifying the dev, attr to modify
val	*string*	The new value this attribute will be assigned to

Outputs

_rv	*integer*	Return status (0 = success)

SYNC adjustClock(int adj)

Changes the speed of the Lifelink/FPGA clock

privilege level ADMIN

Parameters

Inputs

adj	*integer*	A signed value in units of 5.8mHz by which to adjust the clock speed

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getPowerState(enum ll_power_state *state)

Check whether the Lifelink subsystem is currently in a power-related error state

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
state	*unsigned int*	Output value containing the current state

SYNC doEvent(const struct ll_event *event)

Interface invoked by udev to propogate a Lifelink event.

privilege level ADMIN

Parameters

Inputs

event	*structure*	Defines the type of event that occurred, which Lifelink device was involved, and any data pertaining to the event

Outputs

_rv	*integer*	Return status (0 = success)

EVENT devHotplugAdd(ConstStringZ devName)

Event fires when a Lifelink device is connected to the bus

Parameters

devName	*string*	The name of the Lifelink device that has been connected to the bus

EVENT devHotplugRemove(ConstStringZ devName)

Event fires when a Lifelink device is removed from the bus

Parameters

devName	*string*	The name of the Lifelink device that has been removed from the bus

EVENT devButtonDown(ConstStringZ devName, int buttonId)

Event for pressing down a button on a Lifelink device

Parameters

devName	*string*	The device whose button was pressed down
buttonId	*integer*	Which button was pressed down

EVENT devButtonUp(ConstStringZ devName, int buttonId)

Trigger event for releasing a button on a Lifelink device

Parameters

devName	*string*	The device whose pressed button was released
buttonId	*integer*	Which pressed button was released

EVENT powerEvent(enum ll_power_state type)

Power-related Lifelink event

Parameters

type	*unsigned int*	Specifies the type of power-related event

SYNC getVerbose(int *verbose)

Report whether verbose is turned on in the Lifelink driver

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
verbose	*integer*	Will contain true or false, reflecting whether verbose is on or off, respectively

SYNC setVerbose(int verbose)

Turns verbose mode on or off

privilege level ADMIN

Parameters

Inputs

verbose	*integer*	Turns on verbose mode if true, off if false

Outputs

_rv	*integer*	Return status (0 = success)

SYNC sendUpdatePacket(ConstStringZ devName, const unsigned char *packet, int packet_size)

Sends a firmware update packet to the Lifelink device over the chill protocol

privilege level ADMIN

Parameters

Inputs

devName	*string*	The name of the Lifelink device
packet	*array of unsigned int*	The packet containing firmware update data

Outputs

_rv	*integer*	Return status (0 = success)

SYNC waitUpdateResp(ConstStringZ devName, unsigned char

`*respData, int respData_size)`

Waits for a reply to a firmware update packet

privilege level ADMIN

Parameters

Inputs

`devName`	*string*	The name of the Lifelink device
`respData_size`	*integer*	The size of the output buffer

Outputs

`_rv`	*integer*	Return status (0 = success)
`respData`	*array of unsigned int*	The chill protocol response to a firmware update packet

`SYNC rebootDev(ConstStringZ devName)`

Reboots the specified Lifelink device

privilege level ADMIN

Parameters

Inputs

`devName`	*string*	the name of the Lifelink device to reboot

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC rebootAllDevs()`

Reboots all devices connected to the master

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)

LifelinkLed Class Documentation

Overview

Functions by Group

Other Functions

SYNC setLedPattern(LedPattern pattern)

Set led output pattern

privilege level USER

Parameters

Inputs

pattern *unsigned int*

Outputs **_rv** *integer*

SYNC getLedPattern(LedPattern *pattern)

Get led output pattern

privilege level USER

Parameters

Outputs

_rv *integer*

pattern *unsigned int*

EVENT ledPatternChanged(LedPattern pattern)

Event that indicates the LED pattern has changed

Parameters

pattern *unsigned int*

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

Local_Directory Class Documentation

Overview

A directory service actor enables access to a database of directory entries. This can be a local database or a remote database. Each database must have a unique name which is used to register the database with the master directory service. This requires that the initialization of the directory server object use the name in the C++ constructor. Example:

```
Directory_server *srv = new Directory_server("skype_directory")
```

Every database is expected to maintain a locally unique id for each entry in the database. This id is always represented as a string within the context of the directory service but may be an integer or some other type in the actual database. Contact info is represented as a JSON structure with the following fields: "id", "firstname", "lastname", "system", "refid", "number", "type", "protocol", and "bandwidth". These fields may be mapped by the directory service to other fields in the actual remote database, or in some cases, synthesized from fields in the remote database (i.e. firstname and lastname could be derived from a single name field, or dial could be a combination of several fields).

Directory services should use the client methods described in "Managing Directory Services" to register with the master server. Only the master should implement the server methods described there.

Functions by Group

Querying the directory database

beginQuery
itemsRemaining
abortQuery
getNext
requestNext
queryResult

Editing the database

saveEntry
saveEntryDone
deleteEntry
deleteEntryDone
clearAllEntries
clearAllEntriesDone
setConfiguration
getConfiguration
configurationChanged
setLogLevel
getLogLevel
logLevelChanged

presenceChanged
entryCreated
entryUpdated
entryDeleted
reload

Managing directory services

registerServer
deleteServer
getServerEnumerator
enumerateNextServer
requestServers

SYNC beginQuery(ConstStringZ query)

Start a directory lookup operation. The query is a pattern string. The query should return all records where the pattern matches the first characters in any word of the fields "firstname", "lastname", or "systemname". Words are defined as contiguous sequences of alphanumeric characters, so a systemname of "CR-AUS-Alamo" should match the query "ala". Queries are case insensitive.

privilege level ADMIN

Parameters

Inputs

query	*string*	Query specification

Outputs

_rv	*integer*	Query handle or negative error code

SYNC itemsRemaining(int handle)

Get the number of result rows remaining in an active query. This may be an approximation, due to additional filtering needed on the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Number of result entries, or negative error code

ASYNC abortQuery(int handle)

Close out a query before fetching all of the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

SYNC getNext(int handle, StringZ buffer, size_t buffer_size)

Retrieve the next result group from a directory query synchronously. Interchangable with requestNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Result status
		0 Success
		-1 End of query
		-EAGAIN Query not ready, try again
		-EINVAL Bad query handle
		-E2BIG Buffer too small
buffer	*string*	Buffer to accept query results. Buffer will be formatted with a JSON array of structures representing entries. Each entry will have the fields requested in the query.

ASYNC requestNext(int handle)

Request the next result group from a directory query asynchronously. Interchangable with getNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

RESPONSE queryResult(int handle, int status, ConstStringZ result)

Response to requestNext containing query results.

Parameters

handle	*integer*	Query handle used in requestNext
status	*integer*	Query status
		0 Success
		-1 End of query

-EINVAL Bad query handle

result *string* A JSON array of structures representing directory entries matching the query.

ASYNC saveEntry(ConstStringZ info)

Create a new entry in the database or update an existing entry. This will normally only be possible for the local directory. If the entry contains an id field that matches an existing entry in the database then this will perform an update, otherwise it will create the entry.

privilege level ADMIN

Parameters

Inputs

info *string* Contact info record in JSON format

RESPONSE saveEntryDone(int result, ConstStringZ info)

Signals the end of a saveEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded
- -EPERM Operation not allowed
- -EINVAL Invalid info record

info *string* JSON data for contact with directory added id field if no id was present in the original entry.

ASYNC deleteEntry(ConstStringZ id)

Delete an entry in the database. This will normally only be possible for the local directory.

privilege level ADMIN

Parameters

Inputs

id *string* Id field of record to delete.

RESPONSE deleteEntryDone(int result)

Signals the end of a deleteEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded

-EPERM	Operation not allowed
-EINVAL	Invalid info record

ASYNC clearAllEntries()

Remove all entries from the directory.

privilege level ADMIN

RESPONSE clearAllEntriesDone()

Signals the end of a clear-all operation.

SYNC setConfiguration(ConstStringZ config)

Configure remote server parameters. The format of the configuration object can be different for each type of directory service. In general it may include a server address, username, password, and base search string. The local directory does not require this command.

privilege level ADMIN

Parameters

Inputs

config	*string*	JSON formatted configuration object

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getConfiguration(StringZ config, int config_size)

Retrieve the current configuration.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error
config	*string*	Buffer to receive current configuration

EVENT configurationChanged(ConstStringZ server, ConstStringZ config)

Issued when the configuration is changed by setConfiguration.

Parameters

server

	string	Name of reconfigured server
config	*string*	New configuration

SYNC setLogLevel(int mask)

Set the logging mask.

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New logging mask

Outputs

_rv	*integer*	0 or negative error code

SYNC getLogLevel(int *mask)

Retrieve the current logging mask

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error code.
mask	*integer*	logging mask

EVENT logLevelChanged(int mask)

Issued when the logging level is changed.

Parameters

mask	*integer*	New logging level

EVENT presenceChanged(ConstStringZ server, ConstStringZ entries)

Issued to indicate a change of presence for one or more entries.

Parameters

server	*string*	Directory service that provides this entry
entries	*string*	JSON array of entries changed

EVENT entryCreated(ConstStringZ server, ConstStringZ info)

issued when a new entry is created in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Created entry

EVENT entryUpdated(ConstStringZ server, ConstStringZ info)

issued when an entry is updated in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Updated entry

EVENT entryDeleted(ConstStringZ server, ConstStringZ info)

issued when an entry is deleted from the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Deleted entry

EVENT reload(ConstStringZ server)

issued when cached entries for this directory should be invalidated.

Parameters

server	*string*	Actor name for this directory LISTEN Directory_requestServers()

SYNC registerServer(ConstStringZ serverName)

Add a directory server to the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of new server

Outputs

_rv	*integer*	Result status

0	Success
negative	Negative error code

SYNC deleteServer(ConstStringZ serverName)

Remove a directory server from the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of server to remove.

Outputs

_rv	*integer*	Result status
	0	Success
	-ENOENT	Server name not found

SYNC getServerEnumerator()

Begin enumeration of directory servers.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextServer(int handle, StringZ buffer, size_t buffer_size)

Get the next set of servers in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getServerEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has one or more server names.
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive server names. Names are space seperated. As many names as will fit in the buffer are returned.

EVENT requestServers()

Sent by the master directory to request that all directory servers register their name.

MP Class Documentation

Overview

ProxyManager creates a proxy connections using the REST interface and exposes those calls to switchboard.

Functions by Group

Managing the Proxy

ASYNC startProxy(ConstStringZ actor, ConstStringZ host, ConstStringZ username, ConstStringZ password)

Starts a proxy connection with a specific actor type and name.

privilege level ADMIN

Parameters

Inputs

actor	*string*	Actor.service name and base class like: Helium.proxy=He
host	*string*	IP address of the remote bridge
username	*string*	Username to log in to the remote bridge
password	*string*	Password to log in to the remote bridge

ASYNC stopProxy(ConstStringZ actor)

Terminate an active proxy actor

privilege level ADMIN

Parameters

Inputs

actor	*string*	Actor name of proxy

`EVENT proxyStopped(ConstStringZ actor)`

Issued when a proxy terminates.

Parameters

`actor`	*string*	Actor name of proxy

`SYNC setEnable(int enabled)`

Enable or disable the bridge proxy.

privilege level ADMIN

Parameters

Inputs

`enabled`	*integer*	Enable flag 0 disable the proxy 1 enable the proxy

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC getEnable(int *enabled)`

Determine if the proxy is enabled.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`enabled`	*integer*	Enable flag 0 Proxy is disabled 1 proxy is enabled

`EVENT enableChanged(int enabled)`

Issued when the bridge proxy enable is changed.

Parameters

`enabled`	*integer*	**UNDOCUMENTED**

Manager Class Documentation

Overview

The Manager daemon provides a way for an external entity to push schedule and meeting information to a codec.

Functions by Group

Saving and Deleting agenda information

saveEntries
deleteEntries
entriesCreated
entriesUpdated
entriesDeleted
getEntry
getEntries
getNext
requestNext

Launching agenda items

requestEntryLaunch
requestEntryLaunchDone
launchEntry

Enabling and disabling the Manager interface.

getManagerEnabled
setManagerEnabled
managerEnableChanged
getOfflineMode
setOfflineMode
offlineModeChanged

SYNC saveEntries(ConstStringZ jsonArrayOfEntries)

Saves 1 or more entries. If an entry exists, it is updated, if not, it is created. Posts entriesCreated() and/or entriesUpdated() as dictated by the data.

privilege level ADMIN

Parameters

Inputs

jsonArrayOfEntries	*string*	A JSON array of 1 or more entries to save

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteEntries(ConstStringZ jsonArrayOfIds)

Deletes 1 or more entries related to the provide IDs. Posts entriesDeleted() for all the deleted IDs

privilege level ADMIN

Parameters

Inputs

jsonArrayOfIds	*string*	A JSON array of 1 or more entry IDs to delete

Outputs

_rv	*integer*	Return status (0 = success)

EVENT entriesCreated(ConstStringZ jsonArrayOfIds)

posted when 1 or more entries are created

Parameters

jsonArrayOfIds	*string*	A JSON array of entry IDs that were just created

EVENT entriesUpdated(ConstStringZ jsonArrayOfIds)

posted when 1 or more entries are updated

Parameters

jsonArrayOfIds	*string*	A JSON array of entry IDs that were just updated

EVENT entriesDeleted(ConstStringZ jsonArrayOfIds)

posted when 1 or more entries are deleted

Parameters

jsonArrayOfIds	*string*	A JSON array of entry IDs that were just deleted

SYNC getEntry(ConstStringZ id, StringZ buffer, size_t buffer_size)

get the contents of a single entry

privilege level ADMIN

Parameters

Inputs

id	*string*	Entry ID to retrieve

Outputs

_rv	*integer*	Return status (0 = success)
buffer	*string*	Buffer to receive JSON entry

SYNC getEntries()

Returns an iterator used to retrieve all the entries

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	>=0 the iterator, <= error

SYNC getNext(int handle, StringZ buffer, size_t buffer_size)

Returns the next buffer of entries as a JSON array

privilege level ADMIN

Parameters

Inputs

handle	*integer*	The iterator returned by getEntries()

Outputs

_rv	*integer*	Return status (0 = success)
buffer	*string*	The return buffer to hold the JSON array

ASYNC requestNext(int handle)

Returns the next buffer of entries as a JSON array

privilege level ADMIN

Parameters

Inputs

handle	*integer*	The iterator returned by getEntries()

ASYNC requestEntryLaunch(ConstStringZ entryIdToLaunch)

Launches the entry associated with entryIdToLaunch If the entry is a scheduled meeting, the launchEntry() event is posted to notify Manager to launch the meeting. Other meeting types (like bridge meetings than need to be dialed directly) return an error.

privilege level ADMIN

Parameters

Inputs

`entryIdToLaunch`	*string*	The ID of the entry to launch

`RESPONSE requestEntryLaunchDone(int returnValue)`

posted when the delete is complete

Parameters

`returnValue`	*integer*	0==success <0 failure (errors TBD)

`EVENT launchEntry(ConstStringZ entryIdToLaunch)`

posted when a entry is to be launch by Manager

Parameters

`entryIdToLaunch`	*string*	The ID of the entry to launch

`SYNC getManagerEnabled(int* enabled)`

Get Manager enable flag

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`enabled`	*integer*	enable flag
	0=disabled	manager connections are not allowed
	1=enabled	manager connections are allowed

`SYNC setManagerEnabled(int enabled)`

Set Manager enable flag

privilege level ADMIN

Parameters

Inputs

`enabled`	*integer*	enable flag
	0=disabled	manager connections are not allowed
	1=enabled	manager connections are allowed

Outputs

`_rv`		

integer Return status (0 = success)

EVENT managerEnableChanged(int enabled)

Issued when manager enable is changed.

Parameters

enabled *integer* enable flag

0=disabled	manager connections are not allowed
1=enabled	manager connections are allowed

SYNC getOfflineMode(int* enabled)

Get state of offline mode. When the system is offline no calls are allowed to be made.

privilege level ADMIN

Parameters

Outputs

_rv *integer* Return status (0 = success)

enabled *integer* offline mode switch

0=disabled	- offline mode disabled, calls are permitted
1=enabled	- offline mode enabled, no calls are permitted

SYNC setOfflineMode(int enabled)

Set state of offline mode.

privilege level ADMIN

Parameters

Inputs

enabled *integer* offline mode switch

0=disabled	- offline mode disabled, calls are permitted
1=enabled	- offline mode enabled, no calls are permitted

Outputs

_rv *integer* Return status (0 = success)

EVENT offlineModeChanged(int enabled)

Issued whenever the offline mode flag is changed.

Parameters

`enabled`	*integer*	offline mode switch 0=disabled - offline mode disabled, calls are permitted 1=enabled - offline mode enabled, no calls are permitted

MetaDaemon Class Documentation

Overview

Functions by Group

Other Functions

daemonActive

EVENT daemonActive(metadaemon_type_t type, int id, int active)

Called by hardware camera daemons (hdmi_camera/visca_camera/usb_camera/etc) when a camera is ready to be configured

Parameters

type	*unsigned int*	- type of daemon
id	*integer*	- interface id for the daemon (specific to the daemon type)
active	*integer*	- 1 if the daemon is active, 0 if inactive

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

MsMmcpv Class Documentation

Overview

Functions by Group

Other Functions

EVENT IFrame_Received(int callHandle, int bIsAncillaryVideo)

event indicating the reception of intra frame. Event thrown for both main and presentation streams

Parameters

callHandle	*integer*	indicates the call session for which the Iframe was received
bIsAncillaryVideo	*integer*	indicates if its main/presentation stream. 0 indicates main video and 1 indicates presentation stream

SYNC setHWCryptoEnable(int enable)

This API is used to enable or disbale Hardware based NSS-crypto support in commRTP driver. To enable Hardware crypto use MsMmcpv_setHWCryptoEnable(1) and to disable Hardware crypto use MsMmcpv_setHWCryptoEnable(0)

privilege level ADMIN

Parameters

Inputs

enable	*integer*	1 - enable and 0 - disable

Outputs

_rv	*integer*	It returns 0 on success and negative value on failure

SYNC enableCommDebugLog(int mask)

This API is called to enable/disable Comm Debug log

privilege level ADMIN

Parameters

Inputs

mask	*integer*	1 - enable CommDebugLog and 0 - disable CommDebugLog

Outputs

_rv	*integer*	It returns 0 on success and negative value on failure

PMan Class Documentation

Overview

ProxyManager creates a proxy connection to a bridge using the REST interface and exposes those calls to switchboard.

Functions by Group

Managing the Proxy

SYNC setConfig(ConstStringZ host, ConstStringZ username, ConstStringZ password)

Sets the connection parameters for the bridge. Any existing connection is torn down.

privilege level ADMIN

Parameters

Inputs

host	*string*	IP address of the remote bridge
username	*string*	Username to log in to the remote bridge
password	*string*	Password to log in to the remote bridge

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getConfig(StringZ host, size_t host_size, StringZ username, size_t username_size, StringZ password, size_t password_size)

Retrieves the parameters used to connect the proxy to a bridge.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
host	*string*	IP address of the remote bridge
username	*string*	Username to log in to the remote bridge
password	*string*	Password to log in to the remote bridge

EVENT configChanged()

Issued when the bridge proxy parameters are changed.

ASYNC connect()

Attempt a connection to a remote bridge using the current configuration. If a connection is already in place this will do nothing.

privilege level ADMIN

EVENT connectDone(int status)

Issued when a proxy connection attempt completes.

Parameters

status	*integer*	Connection status
	0	Connection Successful
	-1	Connection failed

SYNC setEnable(int enabled)

Enable or disable the bridge proxy.

privilege level ADMIN

Parameters

Inputs

enabled	*integer*	Enable flag
	0	disable the proxy
	1	enable the proxy

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getEnable(int *enabled)

Determine if the proxy is enabled.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
enabled	*integer*	Enable flag 0 Proxy is disabled 1 proxy is enabled

EVENT enableChanged(int enabled)

Issued when the bridge proxy enable is changed.

Parameters

enabled	*integer*	New state of enable flag

Recents_Directory Class Documentation

Overview

A directory service actor enables access to a database of directory entries. This can be a local database or a remote database. Each database must have a unique name which is used to register the database with the master directory service. This requires that the initialization of the directory server object use the name in the C++ constructor. Example:

```
Directory_server *srv = new Directory_server("skype_directory")
```

Every database is expected to maintain a locally unique id for each entry in the database. This id is always represented as a string within the context of the directory service but may be an integer or some other type in the actual database. Contact info is represented as a JSON structure with the following fields: "id", "firstname", "lastname", "system", "refid", "number", "type", "protocol", and "bandwidth". These fields may be mapped by the directory service to other fields in the actual remote database, or in some cases, synthesized from fields in the remote database (i.e. firstname and lastname could be derived from a single name field, or dial could be a combination of several fields).

Directory services should use the client methods described in "Managing Directory Services" to register with the master server. Only the master should implement the server methods described there.

Functions by Group

Querying the directory database

beginQuery
itemsRemaining
abortQuery
getNext
requestNext
queryResult

Editing the database

saveEntry
saveEntryDone
deleteEntry
deleteEntryDone
clearAllEntries
clearAllEntriesDone
setConfiguration
getConfiguration
configurationChanged
setLogLevel
getLogLevel
logLevelChanged

presenceChanged
entryCreated
entryUpdated
entryDeleted
reload

Managing directory services

registerServer
deleteServer
getServerEnumerator
enumerateNextServer
requestServers

SYNC beginQuery(ConstStringZ query)

Start a directory lookup operation. The query is a pattern string. The query should return all records where the pattern matches the first characters in any word of the fields "firstname", "lastname", or "systemname". Words are defined as contiguous sequences of alphanumeric characters, so a systemname of "CR-AUS-Alamo" should match the query "ala". Queries are case insensitive.

privilege level ADMIN

Parameters

Inputs

query	*string*	Query specification

Outputs

_rv	*integer*	Query handle or negative error code

SYNC itemsRemaining(int handle)

Get the number of result rows remaining in an active query. This may be an approximation, due to additional filtering needed on the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Number of result entries, or negative error code

ASYNC abortQuery(int handle)

Close out a query before fetching all of the result data.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

SYNC getNext(int handle, StringZ buffer, size_t buffer_size)

Retrieve the next result group from a directory query synchronously. Interchangable with requestNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

Outputs

_rv	*integer*	Result status
		0 Success
		-1 End of query
		-EAGAIN Query not ready, try again
		-EINVAL Bad query handle
		-E2BIG Buffer too small
buffer	*string*	Buffer to accept query results. Buffer will be formatted with a JSON array of structures representing entries. Each entry will have the fields requested in the query.

ASYNC requestNext(int handle)

Request the next result group from a directory query asynchronously. Interchangable with getNext in overall functionality.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Query handle from beginQuery()

RESPONSE queryResult(int handle, int status, ConstStringZ result)

Response to requestNext containing query results.

Parameters

handle	*integer*	Query handle used in requestNext
status	*integer*	Query status
		0 Success
		-1 End of query

-EINVAL Bad query handle

result *string* A JSON array of structures representing directory entries matching the query.

ASYNC saveEntry(ConstStringZ info)

Create a new entry in the database or update an existing entry. This will normally only be possible for the local directory. If the entry contains an id field that matches an existing entry in the database then this will perform an update, otherwise it will create the entry.

privilege level ADMIN

Parameters

Inputs

info *string* Contact info record in JSON format

RESPONSE saveEntryDone(int result, ConstStringZ info)

Signals the end of a saveEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded
- -EPERM Operation not allowed
- -EINVAL Invalid info record

info *string* JSON data for contact with directory added id field if no id was present in the original entry.

ASYNC deleteEntry(ConstStringZ id)

Delete an entry in the database. This will normally only be possible for the local directory.

privilege level ADMIN

Parameters

Inputs

id *string* Id field of record to delete.

RESPONSE deleteEntryDone(int result)

Signals the end of a deleteEntry operation.

Parameters

result *integer* Status of operation

- 0 Operation succeeded

-EPERM	Operation not allowed
-EINVAL	Invalid info record

ASYNC clearAllEntries()

Remove all entries from the directory.

privilege level ADMIN

RESPONSE clearAllEntriesDone()

Signals the end of a clear-all operation.

SYNC setConfiguration(ConstStringZ config)

Configure remote server parameters. The format of the configuration object can be different for each type of directory service. In general it may include a server address, username, password, and base search string. The local directory does not require this command.

privilege level ADMIN

Parameters

Inputs

config	*string*	JSON formatted configuration object

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getConfiguration(StringZ config, int config_size)

Retrieve the current configuration.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error
config	*string*	Buffer to receive current configuration

EVENT configurationChanged(ConstStringZ server, ConstStringZ config)

Issued when the configuration is changed by setConfiguration.

Parameters

server

	string	Name of reconfigured server
config	*string*	New configuration

SYNC setLogLevel(int mask)

Set the logging mask.

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New logging mask

Outputs

_rv	*integer*	0 or negative error code

SYNC getLogLevel(int *mask)

Retrieve the current logging mask

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 or negative error code.
mask	*integer*	logging mask

EVENT logLevelChanged(int mask)

Issued when the logging level is changed.

Parameters

mask	*integer*	New logging level

EVENT presenceChanged(ConstStringZ server, ConstStringZ entries)

Issued to indicate a change of presence for one or more entries.

Parameters

server	*string*	Directory service that provides this entry
entries	*string*	JSON array of entries changed

EVENT entryCreated(ConstStringZ server, ConstStringZ info)

issued when a new entry is created in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Created entry

EVENT entryUpdated(ConstStringZ server, ConstStringZ info)

issued when an entry is updated in the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Updated entry

EVENT entryDeleted(ConstStringZ server, ConstStringZ info)

issued when an entry is deleted from the directory.

Parameters

server	*string*	Actor name of the issuing directory service
info	*string*	Deleted entry

EVENT reload(ConstStringZ server)

issued when cached entries for this directory should be invalidated.

Parameters

server	*string*	Actor name for this directory LISTEN Directory_requestServers()

SYNC registerServer(ConstStringZ serverName)

Add a directory server to the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of new server

Outputs

_rv	*integer*	Result status

0	Success
negative	Negative error code

SYNC deleteServer(ConstStringZ serverName)

Remove a directory server from the list of active servers.

privilege level ADMIN

Parameters

Inputs

serverName	*string*	Name of server to remove.

Outputs

_rv	*integer*	Result status
	0	Success
	-ENOENT	Server name not found

SYNC getServerEnumerator()

Begin enumeration of directory servers.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextServer(int handle, StringZ buffer, size_t buffer_size)

Get the next set of servers in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getServerEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has one or more server names.
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive server names. Names are space seperated. As many names as will fit in the buffer are returned.

EVENT requestServers()

Sent by the master directory to request that all directory servers register their name.

Remote Class Documentation

Overview

Functions by Group

Other Functions

keyEvent

`EVENT keyEvent(key_event_t event, key_code_t code)`

Notification to the UI that a IR/remote key event has been received

Parameters

`event`	*unsigned int*	- type of key event
`code`	*unsigned int*	- key-code id

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

SB Class Documentation

Overview

Functions by Group

Other Functions

SYNC listen(eventName);

listen for events or responses.

privilege level USER

Parameters

Inputs

eventName	*string*	event to listen for

Outputs

_rv	*integer*	Return status (0 = success)

SYNC unlisten();

unused

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

Serial Class Documentation

Overview

A serial actor controls and configures access to a serial devices. A list of serial devices is available via enumerateNextSerialDevice. A list of serial shells is available via enumerateNextSerialShell. The serial shell and serial shell configuration are available using the getShell function. The serial shell and serial configuration can be changed by using the setShell function. Shell or config changes are reported by the shellChanged event.

Functions by Group

shell configuration

SYNC setShell(ConstStringZ shellname, ConstStringZ config)

Set serial device shell and configuration.

privilege level ADMIN

Parameters

Inputs

shellname	*string*	Name of the shell, use enumerateNextSerialShell to get valid values
config	*string*	Shell specific configuration, comma separated

dvi	when shell is visca the DVI input gets associated to this shell
hdmi	when shell is visca the HDMI input gets associated to this shell
1200	when shell is admin the bit rate is 1200 bps
2400	when shell is admin the bit rate is 2400 bps

4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code

SYNC getShell(StringZ shell, size_t shell_size, StringZ config, size_t config_size)

Get the shell on the serial device.

privilege level USER

Parameters

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code
shell	*string*	Name of the shell
	none	No shell
	visca	Visca protocol shell
	admin	Admin user shell
config	*string*	Shell specific configuration, comma separated
	dvi	when shell is visca the DVI input gets associated to this shell
	hdmi	when shell is visca the HDMI input gets associated to this shell
	1200	when shell is admin the bit rate is

	1200 bps
2400	when shell is admin the bit rate is 2400 bps
4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

`EVENT shellChanged(ConstStringZ devActor)`

Notification that a serial device has changed configuration

Parameters

`devActor`	*string*	Name of the actor to query for more information related to this change

`SYNC getSerialDeviceEnumerator()`

Begin enumeration of serial devices.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Enumeration handle or negative error code

`SYNC enumerateNextSerialDevice(int handle, StringZ buffer, size_t buffer_size)`

Get the next set of serial devices in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialDeviceEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial device name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive serial device name.

SYNC getSerialShellEnumerator()

Begin enumeration of serial shells.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextSerialShell(int handle, StringZ shellname, size_t shellname_size, StringZ jsonconfig, size_t jsonconfig_size)

Get the next serial shell and config in an enumeration.

privilege level ADMIN

ERROR: unknown @param config

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialShellEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial shell name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
shellname	*string*	serial shell name.
jsonconfig	*string*	**UNDOCUMENTED**

Serial1 Class Documentation

Overview

A serial actor controls and configures access to a serial devices. A list of serial devices is available via enumerateNextSerialDevice. A list of serial shells is available via enumerateNextSerialShell. The serial shell and serial shell configuration are available using the getShell function. The serial shell and serial configuration can be changed by using the setShell function. Shell or config changes are reported by the shellChanged event.

Functions by Group

shell configuration

setShell
getShell
shellChanged
getSerialDeviceEnumerator
enumerateNextSerialDevice
getSerialShellEnumerator
enumerateNextSerialShell

SYNC setShell(ConstStringZ shellname, ConstStringZ config)

Set serial device shell and configuration.

privilege level ADMIN

Parameters

Inputs

shellname	*string*	Name of the shell, use enumerateNextSerialShell to get valid values
config	*string*	Shell specific configuration, comma separated

dvi	when shell is visca the DVI input gets associated to this shell
hdmi	when shell is visca the HDMI input gets associated to this shell
1200	when shell is admin the bit rate is 1200 bps
2400	when shell is admin the bit rate is 2400 bps

4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code

SYNC getShell(StringZ shell, size_t shell_size, StringZ config, size_t config_size)

Get the shell on the serial device.

privilege level USER

Parameters

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code
shell	*string*	Name of the shell
	none	No shell
	visca	Visca protocol shell
	admin	Admin user shell
config	*string*	Shell specific configuration, comma separated
	dvi	when shell is visca the DVI input gets associated to this shell
	hdmi	when shell is visca the HDMI input gets associated to this shell
	1200	when shell is admin the bit rate is

	1200 bps
2400	when shell is admin the bit rate is 2400 bps
4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

`EVENT shellChanged(ConstStringZ devActor)`

Notification that a serial device has changed configuration

Parameters

`devActor`	*string*	Name of the actor to query for more information related to this change

`SYNC getSerialDeviceEnumerator()`

Begin enumeration of serial devices.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Enumeration handle or negative error code

`SYNC enumerateNextSerialDevice(int handle, StringZ buffer, size_t buffer_size)`

Get the next set of serial devices in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialDeviceEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial device name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive serial device name.

SYNC getSerialShellEnumerator()

Begin enumeration of serial shells.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextSerialShell(int handle, StringZ shellname, size_t shellname_size, StringZ jsonconfig, size_t jsonconfig_size)

Get the next serial shell and config in an enumeration.

privilege level ADMIN

ERROR: unknown @param config

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialShellEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial shell name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
shellname	*string*	serial shell name.
jsonconfig	*string*	**UNDOCUMENTED**

Serial2 Class Documentation

Overview

A serial actor controls and configures access to a serial devices. A list of serial devices is available via enumerateNextSerialDevice. A list of serial shells is available via enumerateNextSerialShell. The serial shell and serial shell configuration are available using the getShell function. The serial shell and serial configuration can be changed by using the setShell function. Shell or config changes are reported by the shellChanged event.

Functions by Group

shell configuration

setShell
getShell
shellChanged
getSerialDeviceEnumerator
enumerateNextSerialDevice
getSerialShellEnumerator
enumerateNextSerialShell

SYNC setShell(ConstStringZ shellname, ConstStringZ config)

Set serial device shell and configuration.

privilege level ADMIN

Parameters

Inputs

shellname	*string*	Name of the shell, use enumerateNextSerialShell to get valid values
config	*string*	Shell specific configuration, comma separated

dvi	when shell is visca the DVI input gets associated to this shell
hdmi	when shell is visca the HDMI input gets associated to this shell
1200	when shell is admin the bit rate is 1200 bps
2400	when shell is admin the bit rate is 2400 bps

4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code

SYNC getShell(StringZ shell, size_t shell_size, StringZ config, size_t config_size)

Get the shell on the serial device.

privilege level USER

Parameters

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code
shell	*string*	Name of the shell
	none	No shell
	visca	Visca protocol shell
	admin	Admin user shell
config	*string*	Shell specific configuration, comma separated
	dvi	when shell is visca the DVI input gets associated to this shell
	hdmi	when shell is visca the HDMI input gets associated to this shell
	1200	when shell is admin the bit rate is

	1200 bps
2400	when shell is admin the bit rate is 2400 bps
4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

`EVENT shellChanged(ConstStringZ devActor)`

Notification that a serial device has changed configuration

Parameters

`devActor`	*string*	Name of the actor to query for more information related to this change

`SYNC getSerialDeviceEnumerator()`

Begin enumeration of serial devices.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Enumeration handle or negative error code

`SYNC enumerateNextSerialDevice(int handle, StringZ buffer, size_t buffer_size)`

Get the next set of serial devices in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialDeviceEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial device name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive serial device name.

SYNC getSerialShellEnumerator()

Begin enumeration of serial shells.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextSerialShell(int handle, StringZ shellname, size_t shellname_size, StringZ jsonconfig, size_t jsonconfig_size)

Get the next serial shell and config in an enumeration.

privilege level ADMIN

ERROR: unknown @param config

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialShellEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial shell name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
shellname	*string*	serial shell name.
jsonconfig	*string*	**UNDOCUMENTED**

ShellAdmin Class Documentation

Overview

A serial actor controls and configures access to a serial devices. A list of serial devices is available via enumerateNextSerialDevice. A list of serial shells is available via enumerateNextSerialShell. The serial shell and serial shell configuration are available using the getShell function. The serial shell and serial configuration can be changed by using the setShell function. Shell or config changes are reported by the shellChanged event.

Functions by Group

shell configuration

setShell
getShell
shellChanged
getSerialDeviceEnumerator
enumerateNextSerialDevice
getSerialShellEnumerator
enumerateNextSerialShell

SYNC setShell(ConstStringZ shellname, ConstStringZ config)

Set serial device shell and configuration.

privilege level ADMIN

Parameters

Inputs

shellname	*string*	Name of the shell, use enumerateNextSerialShell to get valid values
config	*string*	Shell specific configuration, comma separated

dvi	when shell is visca the DVI input gets associated to this shell
hdmi	when shell is visca the HDMI input gets associated to this shell
1200	when shell is admin the bit rate is 1200 bps
2400	when shell is admin the bit rate is 2400 bps

4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code

SYNC getShell(StringZ shell, size_t shell_size, StringZ config, size_t config_size)

Get the shell on the serial device.

privilege level USER

Parameters

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code
shell	*string*	Name of the shell
	none	No shell
	visca	Visca protocol shell
	admin	Admin user shell
config	*string*	Shell specific configuration, comma separated
	dvi	when shell is visca the DVI input gets associated to this shell
	hdmi	when shell is visca the HDMI input gets associated to this shell
	1200	when shell is admin the bit rate is

	1200 bps
2400	when shell is admin the bit rate is 2400 bps
4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

`EVENT shellChanged(ConstStringZ devActor)`

Notification that a serial device has changed configuration

Parameters

`devActor`	*string*	Name of the actor to query for more information related to this change

`SYNC getSerialDeviceEnumerator()`

Begin enumeration of serial devices.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Enumeration handle or negative error code

`SYNC enumerateNextSerialDevice(int handle, StringZ buffer, size_t buffer_size)`

Get the next set of serial devices in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialDeviceEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial device name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive serial device name.

SYNC getSerialShellEnumerator()

Begin enumeration of serial shells.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextSerialShell(int handle, StringZ shellname, size_t shellname_size, StringZ jsonconfig, size_t jsonconfig_size)

Get the next serial shell and config in an enumeration.

privilege level ADMIN

ERROR: unknown @param config

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialShellEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial shell name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
shellname	*string*	serial shell name.
jsonconfig	*string*	**UNDOCUMENTED**

ShellNone Class Documentation

Overview

A serial actor controls and configures access to a serial devices. A list of serial devices is available via enumerateNextSerialDevice. A list of serial shells is available via enumerateNextSerialShell. The serial shell and serial shell configuration are available using the getShell function. The serial shell and serial configuration can be changed by using the setShell function. Shell or config changes are reported by the shellChanged event.

Functions by Group

shell configuration

setShell
getShell
shellChanged
getSerialDeviceEnumerator
enumerateNextSerialDevice
getSerialShellEnumerator
enumerateNextSerialShell

SYNC setShell(ConstStringZ shellname, ConstStringZ config)

Set serial device shell and configuration.

privilege level ADMIN

Parameters

Inputs

shellname	*string*	Name of the shell, use enumerateNextSerialShell to get valid values
config	*string*	Shell specific configuration, comma separated

dvi	when shell is visca the DVI input gets associated to this shell
hdmi	when shell is visca the HDMI input gets associated to this shell
1200	when shell is admin the bit rate is 1200 bps
2400	when shell is admin the bit rate is 2400 bps

4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code

SYNC getShell(StringZ shell, size_t shell_size, StringZ config, size_t config_size)

Get the shell on the serial device.

privilege level USER

Parameters

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code
shell	*string*	Name of the shell
	none	No shell
	visca	Visca protocol shell
	admin	Admin user shell
config	*string*	Shell specific configuration, comma separated
	dvi	when shell is visca the DVI input gets associated to this shell
	hdmi	when shell is visca the HDMI input gets associated to this shell
	1200	when shell is admin the bit rate is

	1200 bps
2400	when shell is admin the bit rate is 2400 bps
4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

`EVENT shellChanged(ConstStringZ devActor)`

Notification that a serial device has changed configuration

Parameters

`devActor`	*string*	Name of the actor to query for more information related to this change

`SYNC getSerialDeviceEnumerator()`

Begin enumeration of serial devices.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Enumeration handle or negative error code

`SYNC enumerateNextSerialDevice(int handle, StringZ buffer, size_t buffer_size)`

Get the next set of serial devices in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialDeviceEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial device name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive serial device name.

SYNC getSerialShellEnumerator()

Begin enumeration of serial shells.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextSerialShell(int handle, StringZ shellname, size_t shellname_size, StringZ jsonconfig, size_t jsonconfig_size)

Get the next serial shell and config in an enumeration.

privilege level ADMIN

ERROR: unknown @param config

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialShellEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial shell name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
shellname	*string*	serial shell name.
jsonconfig	*string*	**UNDOCUMENTED**

ShellVisca Class Documentation

Overview

A serial actor controls and configures access to a serial devices. A list of serial devices is available via enumerateNextSerialDevice. A list of serial shells is available via enumerateNextSerialShell. The serial shell and serial shell configuration are available using the getShell function. The serial shell and serial configuration can be changed by using the setShell function. Shell or config changes are reported by the shellChanged event.

Functions by Group

shell configuration

SYNC setShell(ConstStringZ shellname, ConstStringZ config)

Set serial device shell and configuration.

privilege level ADMIN

Parameters

Inputs

shellname	*string*	Name of the shell, use enumerateNextSerialShell to get valid values
config	*string*	Shell specific configuration, comma separated

dvi	when shell is visca the DVI input gets associated to this shell
hdmi	when shell is visca the HDMI input gets associated to this shell
1200	when shell is admin the bit rate is 1200 bps
2400	when shell is admin the bit rate is 2400 bps

4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code

SYNC getShell(StringZ shell, size_t shell_size, StringZ config, size_t config_size)

Get the shell on the serial device.

privilege level USER

Parameters

Outputs

_rv	*integer*	Result status
	0	Success
	negative	Negative error code
shell	*string*	Name of the shell
	none	No shell
	visca	Visca protocol shell
	admin	Admin user shell
config	*string*	Shell specific configuration, comma separated
	dvi	when shell is visca the DVI input gets associated to this shell
	hdmi	when shell is visca the HDMI input gets associated to this shell
	1200	when shell is admin the bit rate is

	1200 bps
2400	when shell is admin the bit rate is 2400 bps
4800	when shell is admin the bit rate is 4800 bps
9600	when shell is admin the bit rate is 9600 bps
19200	when shell is admin the bit rate is 19200 bps
38400	when shell is admin the bit rate is 38400 bps
57600	when shell is admin the bit rate is 57600 bps
115200	when shell is admin the bit rate is 115200 bps (default)
noflow	when shell is admin indicates no flow control
hardwareflow	when shell is admin indicates hardware flow control
softwareflow	when shell is admin indicates software flow control

`EVENT shellChanged(ConstStringZ devActor)`

Notification that a serial device has changed configuration

Parameters

`devActor`	*string*	Name of the actor to query for more information related to this change

`SYNC getSerialDeviceEnumerator()`

Begin enumeration of serial devices.

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Enumeration handle or negative error code

`SYNC enumerateNextSerialDevice(int handle, StringZ buffer, size_t buffer_size)`

Get the next set of serial devices in an enumeration.

privilege level ADMIN

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialDeviceEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial device name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
buffer	*string*	Buffer to receive serial device name.

SYNC getSerialShellEnumerator()

Begin enumeration of serial shells.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Enumeration handle or negative error code

SYNC enumerateNextSerialShell(int handle, StringZ shellname, size_t shellname_size, StringZ jsonconfig, size_t jsonconfig_size)

Get the next serial shell and config in an enumeration.

privilege level ADMIN

ERROR: unknown @param config

Parameters

Inputs

handle	*integer*	Enumeration handle from getSerialShellEnumerator

Outputs

_rv	*integer*	Result status
	0	Success, buffer has serial shell name
	-1	End of enumeration
	-EINVAL	Bad enumeration handle
shellname	*string*	serial shell name.
jsonconfig	*string*	**UNDOCUMENTED**

SysAdmin Class Documentation

Overview

SysAdmin provides system administration and network configuration services.

Functions by Group

General capabilities

Communications port management

Managing interface Link state

IPv4 Configuration

External stimulus and logging

System services

SYNC getAesCapable()

determine if the system is AES capable or not

privilege level USER

Parameters

Outputs

_rv	*integer*	0 = Not capable, 1 = Capable

SYNC getNumberOfInterfaces()

Retrieve the number of network connections available

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Number of available network interfaces

SYNC setHostname(ConstStringZ hostname)

sets the system hostname

privilege level ADMIN

Parameters

Inputs

hostname	*string*	**UNDOCUMENTED**

Outputs

_rv	*integer*	Hostname

SYNC getHostname(StringZ hostname, size_t hostname_size)

gets the system hostname

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Hostname
hostname	*string*	**UNDOCUMENTED**

SYNC setActivePort(int ifnum)

Set the port to be used by Comm.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Port to be used for VC communications.

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getActivePort()

Get the port used for video conferencing.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Active port or -1 if no port is selected.

EVENT activePortChanged(int ifnum)

Issued when the active port changes

Parameters

ifnum	*integer*	New active port

SYNC getLinkState(int ifnum, int *state)

Gets the link state for a given interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
state	*integer*	Current Link State

NoCarrier=0	Link NoCarrier
Binding=1	Link Binding
Connected=2	Link Connected
NoAddress=3	Link NoAddress
Bonded=4	Link Bonded

EVENT linkStateChanged(int ifnum, int state)

Notification of change of link state

Parameters

ifnum	*integer*	Interface that has changed state
state	*integer*	

Down=0	Link went down
Up=1	Link came up

SYNC setLinkSpeed(int ifnum, int linkSpeed)

Configure speed of an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to configure
linkSpeed	*integer*	Speed to configure (10, 100, 1000)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLinkSpeed(int ifnum, int *linkSpeed)

Retrieve the current link speed

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
linkSpeed	*integer*	Returned speed of interface (10, 100, 1000)

EVENT linkSpeedChanged(int ifnum, int linkSpeed)

Notification of a change in link speed on an interface

Parameters

ifnum	*integer*	Interface whose speed has changed
linkSpeed	*integer*	New speed of interface

SYNC setLinkDuplex(int ifnum, int linkDuplex)

Set duplex of a link

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to change
linkDuplex	*integer*	New duplex mode of link
		Half=0 Half Duplex
		Full=1 Full Duplex

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLinkDuplex(int ifnum, int *linkDuplex)

get the current duplex mode of a link

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
linkDuplex	*integer*	Current link duplex mode
		Half=0 Half Duplex
		Full=1 Full Duplex

EVENT linkDuplexChanged(int ifnum, int linkDuplex)

Notification of link Duplex change

Parameters

ifnum	*integer*	Interface that has changed.
linkDuplex	*integer*	New duplex mode of link
		Half=0 Half Duplex
		Full=1 Full Duplex

SYNC setLinkAutoNeg(int ifnum, int linkAutoNeg)

Enable or disable autonegotiation of link mode.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to change.
linkAutoNeg	*integer*	Disabled=0 Disable autonegotiation
		Enabled=1 Enable autonegotiation

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLinkAutoNeg(int ifnum, int *linkAutoNeg)

retrieve autonegotiation capability.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query.

Outputs

_rv	*integer*	Return status (0 = success)
linkAutoNeg	*integer*	Current autonegotiation status
		Disabled=0 Disable autonegotiation
		Enabled=1 Enable autonegotiation

EVENT linkAutoNegChanged(int ifnum, int linkAutoNeg)

Notification of change in link autonegotiation enable.

Parameters

ifnum	*integer*	Interface that has changed.
linkAutoNeg	*integer*	New autonegotiation status

Disabled=0	Disable autonegotiation
Enabled=1	Enable autonegotiation

SYNC setIPv4StaticConfig(int ifnum, int dhcp, ConstStringZ address, ConstStringZ netmask, ConstStringZ broadcast)

Set IPv4 configuration of an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to configure.
dhcp	*integer*	0 = use static configuration, 1 = use DHCP for configuration
address	*string*	Static IP address (must be valid when dhcp=0, ignored when dhcp=1)
netmask	*string*	Static IP netmask (must be valid when dhcp=0, ignored when dhcp=1)
broadcast	*string*	Static broadcast address (must be valid when dhcp=0, tested when dhcp=1)

Outputs

_rv	*integer*	Return status

0	success
- EINVAL	address, netmask, or broadcast are not valid ip addresses or NULL or empty

SYNC getIPv4StaticConfig(int ifnum, int *dhcp, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Get the current IPv4 configuration for an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
dhcp	*integer*	

Static=0	Static IP configuration
DHCP=1	use DHCP configuration

address	*string*	Static IP address (empty if dhcp == 1)
netmask	*string*	Static netmask (empty if dhcp == 1)
broadcast	*string*	Static broadcast address (empty if dhcp == 1)

EVENT ipv4StaticConfigUpdated()

Notification that the IPv4 configuration changed for an interface

SYNC getIPv4Config(int ifnum, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Retrieve the active IP configuration for an interface, If the interface is configured with DHCP then this
call will return the allocated network configuration

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to query

Outputs

_rv	*integer*	Return status (0 = success)
address	*string*	IP address of interface
netmask	*string*	netmask of interface
broadcast	*string*	broadcast address of interface

SYNC getActiveIPv4Config(int *ifnum, StringZ address, size_t address_size, StringZ netmask, size_t netmask_size, StringZ broadcast, size_t broadcast_size)

Retrieve the active IP configuration for Helium. This is bond0 if bonding is enabled, otherwise, it is the first valid ip address.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
ifnum	*integer*	of Active interface.
address	*string*	IP address of interface
netmask	*string*	netmask of interface
broadcast	*string*	broadcast address of interface

EVENT ipv4ConfigUpdated()

Notification that the active configuration of a network interface has changed

SYNC setIPv6StaticConfig(int ifnum, int enable, int autoconfig, ConstStringZ address, ConstStringZ gateway)

Set the IPv6 configuration of an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to configure	
enable	*integer*	Disable=0	disable IPv6 for this interface
		Enable=1	enable IPv6 for this interface
autoconfig	*integer*	Manual=0	use manual IPv6 address assignment
		Auto=1	use automatic IPv6 address assignment
address	*string*	manually assigned IPv6 address (ignored if automatic address configuration enabled)	
gateway	*string*	IPv6 gateway address (ignored if automatic address configuration enabled)	

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getIPv6Config(int ifnum, int *enable, int *autoconfig, StringZ address, int address_size, StringZ gateway, int gateway_size)

Retrieve the current configuration for an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to query	

Outputs

_rv	*integer*	Return status (0 = success)	
enable	*integer*	Disabled=0	IPv6 is disabled for this interface
		Enabled=1	IPv6 is enabled for this interface
autoconfig	*integer*	Manual=0	this device uses manual IPv6 address assignment
		Auto=1	this device uses automatic IPv6 address assignment
address	*string*	currently assigned IPv6 address	
gateway	*string*	current IPv6 gateway address	

EVENT ipv6StaticConfigChanged(int ifnum)

Issued when a new IPv6 configuration is set using setIPv6StaticConfig.

Parameters

ifnum	*integer*	Interface with new configuration.

EVENT ipv6ConfigChanged(int ifnum)

Issued when the active IPv6 address or gateway address changes for an interface.

Parameters

ifnum	*integer*	Interface with new active configuration.

SYNC addIPv4AliasAddress(int ifnum, ConstStringZ address, ConstStringZ netmask, ConstStringZ broadcast)

create an IP alias on an ethernet interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to add address to
address	*string*	New alias address
netmask	*string*	New alias netmask
broadcast	*string*	New alias broadcast address

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteIPv4AliasAddress(int ifnum, ConstStringZ address)

Remove an IP alias from an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to remove address from
address	*string*	Alias address to remove

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getAliasEnumerator(int ifnum)

Start an enumeration of aliases for an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to enumerate

Outputs

_rv	*integer*	Enumerator ID

SYNC enumerateNextAlias(int id, StringZ address, int address_size)

Retrieve the next alias address in an enumeration

privilege level ADMIN

Parameters

Inputs

id	*integer*	Enumerator ID from getAliasEnumerator()

Outputs

_rv	*integer*	status 0 address contains a valid alias address -1 end of enumeration
address	*string*	Returned alias address

SYNC addIPv6AliasAddress(int ifnum, ConstStringZ address)

Add an IPv6 address to an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to add address to
address	*string*	IPv6 address with prefix length in slash notation

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteIPv6AliasAddress(int ifnum, ConstStringZ address)

remove an IPv6 address from an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to remove address from
address	*string*	address to remove with prefix length

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getIPv6AliasEnumerator(int ifnum)

get an enumerator id for the IPv6 address aliases on an interface.

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to enumerate

Outputs

_rv	*integer*	enumerator id to be used in enumerateNextIPv6Alias or negative error

SYNC enumerateNextIPv6Alias(int id, StringZ address, int address_size)

Get the next address in an enumeration.

privilege level ADMIN

Parameters

Inputs

id	*integer*	enumerator id

Outputs

_rv	*integer*	status 0 buffer contains a valid IPv6 address and prefix length - end of enumeration or invalid enumerator 1
address	*string*	buffer to receive IPv6 address

SYNC setVlan(int ifnum, int vlan)

set the VLAN to be used on an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to set
vlan	*integer*	VLAN identifier 0-4094

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getVlan(int ifnum, int *vlan)

get the VLAN setting for an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	interface to query

Outputs

_rv	*integer*	Return status (0 = success)
vlan	*integer*	VLAN identifier 0-4094

EVENT vlanChanged(int ifnum,int vlanId)

issued when the VLAN configuration of an interface is changed

Parameters

ifnum	*integer*	interface that changed
vlanId	*integer*	VLAN identifier 0-4094

SYNC setIPv4StaticGateway(ConstStringZ gateway)

Sets the IPv4 gateway to be used for static network configuration

privilege level ADMIN

Parameters

Inputs

gateway	*string*	IP address of the default gateway

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getIPv4StaticGateway(StringZ gateway, size_t gateway_size)

Retrieve the current static gateway configuration

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
gateway	*string*	IP address of default gateway

EVENT ipv4StaticGatewayUpdated()

Notification of a change in static gateway configuration

SYNC getIPv4Gateway(StringZ gateway, size_t gateway_size)

Retrieve the active IPv4 gateway

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
gateway	*string*	IP address of the current active default gateway

EVENT ipv4GatewayUpdated()

Notification of a change in default IPv4 gateway

SYNC addRoute(ConstStringZ family, ConstStringZ net, ConstStringZ netmask, ConstStringZ gateway, int metric, int mss, ConstStringZ interface)

Create a new route entry

privilege level ADMIN

Parameters

Inputs

family	*string*	address family IPv4=0 IPv4 IPv6=1 IPv6
net	*string*	address
netmask	*string*	network address mask (0 if address is a host)
gateway	*string*	gateway IP address (empty string if no gateway)
metric	*integer*	metric field (0 to omit)
mss	*integer*	max segment size for this route (0 == default)
interface	*string*	interface to be used for this route (empty string to omit)

Outputs

_rv	*integer*	Return status (0 = success)

SYNC deleteRoute(ConstStringZ family, ConstStringZ net,

ConstStringZ netmask)

Delete a route entry

privilege level ADMIN

Parameters

Inputs

family	*string*	address family IPv4=0 IPv4 IPv6=1 IPv6
net	*string*	address
netmask	*string*	network address mask (0 if address is a host)

Outputs

_rv	*integer*	Return status (0 = success)

EVENT routeChanged()

issued when the routing table is changed

SYNC getRouteEnumerator()

start an enumeration of the routing table

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	enumerator or negative error indication

SYNC enumerateNextRoute(int id, StringZ route, int route_size)

returns an entry in the routing table in csv format
field order is addr, mask, gateway, flags, metric, interface.

privilege level ADMIN

Parameters

Inputs

id	*integer*	enumerator id

Outputs

_rv	*integer*	Return status (0 = success)
route	*string*	buffer to receive route description

SYNC setStaticNTPServer(ConstStringZ ntpserver)

set the NTP server to use if there is no DHCP supplied server

privilege level ADMIN

Parameters

Inputs

`ntpserver`	*string*	IP address of an NTP server

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC getStaticNTPServer(StringZ ntpserver, size_t ntpserver_size)`

Retrieve the current statically configured NTP server

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`ntpserver`	*string*	IP address of the NTP server

`EVENT staticNTPServerUpdated()`

Notification of a change in static NTP server configuration

`SYNC getNTPServer(StringZ ntpserver, size_t ntpserver_size)`

Retrieve the currently active NTP server

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`ntpserver`	*string*	IP address of the NTP server

`EVENT ntpServerUpdated()`

Notification of a change in active NTP server

`SYNC setStaticDNSServers(ConstStringZ dnsServers)`

Set a list of DNS servers to use

privilege level ADMIN

Parameters

Inputs

dnsServers	*string*	Space separated list of IP addresses of DNS servers

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getStaticDNSServers(StringZ dnsServers, size_t dnsServers_size)

Retrieve the currently configured list of DNS servers

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsServers	*string*	Space separated list of DNS servers

EVENT staticDNSServersUpdated()

Notification that the configured list of DNS servers has changed

SYNC getDNSServers(StringZ dnsServers, size_t dnsServers_size)

Retrieve the active list of DNS servers

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsServers	*string*	Space separated list of DNS server IP addresses

EVENT dnsServersUpdated()

Notification that the active list of DNS servers has changed

SYNC setStaticDNSDomain(ConstStringZ dnsDomain)

Set the DNS domain to use if none is supplied by DHCP

privilege level ADMIN

Parameters

Inputs

dnsDomain	*string*	DNS domain string

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getStaticDNSDomain(StringZ dnsDomain, size_t dnsDomain_size)

Retrieve the configured static DNS domain

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsDomain	*string*	Configured DNS domain name

EVENT staticDNSDomainUpdated()

Notification that the configuration for DNS domain has changed

SYNC getDNSDomain(StringZ dnsDomain, size_t dnsDomain_size)

Retrieve the active DNS domain name

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsDomain	*string*	Active DNS domain name

EVENT dnsDomainUpdated()

Notification that the active DNS domain has changed

SYNC setStaticDNSSearch(ConstStringZ dnsSearch)

Set list of DNS domains to search

privilege level ADMIN

Parameters

Inputs

dnsSearch	*string*	Space separated list of DNS domains

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getStaticDNSSearch(StringZ dnsSearch, size_t dnsSearch_size)

Retrieve current static configuration of DNS search list

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsSearch	*string*	Space separated list of DNS domains

EVENT staticDNSSearchUpdated()

Notification of change to static DNS search list configuration

SYNC getDNSSearch(StringZ dnsSearch, size_t dnsSearch_size)

Retrieve the currently active DNS search list

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
dnsSearch	*string*	Space separated list of DNS domains

EVENT dnsSearchUpdated()

Notification that active DNS search list has changed

SYNC setInterfaceEnable(int ifnum, int enable)

Enable or disable an interface

privilege level ADMIN

Parameters

Inputs

ifnum	*integer*	Interface to configure	
enable	*integer*	Disabled=0	Disable interface

Enabled=1 Enable interface

Outputs

_rv *integer* Return status (0 = success)

SYNC getInterfaceEnable(int ifnum, int *enable)

Retrieve the current enable status of an interface

privilege level ADMIN

Parameters

Inputs

ifnum *integer* Interface to query

Outputs

_rv *integer* Return status (0 = success)

enable *integer* Disabled=0 Disable interface

Enabled=1 Enable interface

EVENT interfaceEnableUpdated(int ifnum)

Notification of a change in enable status for an interface

Parameters

ifnum *integer* Interface that changed

SYNC changePassword(ConstStringZ username, ConstStringZ password)

Change the password for a SOAP user

privilege level ADMIN

Parameters

Inputs

username *string* Name of user to change

password *string* New password for user

Outputs

_rv *integer* Return status (0 = success)

EVENT passwordChanged(ConstStringZ username)

Issued when a password is changed.

Parameters

username *string* user name that password was changed for

SYNC setTime(time_t utcTime)

Set the current system time

privilege level ADMIN

Parameters

Inputs

utcTime	*integer*	UTC time in seconds from the epoch

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getTime(time_t *utcTime)

Retrieve the current system time

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
utcTime	*integer*	Current system time in seconds since the epoch

EVENT timeUpdated()

Notification that system time has changed due to setTime

SYNC setTimeZone(ConstStringZ timezone)

Set the system time zone

privilege level ADMIN

Parameters

Inputs

timezone	*string*	A time zone string such as "America/Chicago"

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getTimeZone(StringZ timezone, size_t timezone_size)

Retrieve the current system time zone

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
timezone	*string*	Current system time zone string

EVENT timeZoneUpdated()

Notification that the system time zone has been changed

SYNC getTimeZoneOffset(ConstStringZ timezone, int* offset)

Retrieve the offset in seconds from GMT of the named timezone

privilege level ADMIN

Parameters

Inputs

timezone	*string*	Timezone to calculate

Outputs

_rv	*integer*	Return status (0 = success)
offset	*integer*	Offset of time zone in seconds west of GMT

ASYNC adminEvent(ConstStringZ data)

Send an external command to SysAdmin through a script

privilege level ADMIN

Parameters

Inputs

data	*string*	Command text

SYNC setLogLevel(int mask)

Set the log level for the sysadmin service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

returns the current log level mask of sysadmin

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`mask`	*integer*	returned mask

`EVENT logLevelChanged(int mask)`

issued when the sysadmin log level changes

Parameters

`mask`	*integer*	new log level mask

`EVENT maintenanceConfigUpdated( )`

Notification that the maintenence tunnel configuration has changed

`SYNC lockReset( )`

lock the system from doing a reset until lock is released

privilege level USER

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC resetIsLocked( )`

is the reset lock on

privilege level USER

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC unlockReset( )`

un lock the system from doing a reset until lock is released

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)

SYNC reboot(ConstStringZ reason)

Request a system reboot

privilege level USER

Parameters

Inputs

reason	*string*	String to be placed in the reset log

Outputs

_rv	*integer*	Return status (0 = success)

SYNC reset(ConstStringZ reason, int options)

Request a system reset with options

privilege level ADMIN

Parameters

Inputs

reason	*string*	String to be appended to reset log
options	*integer*	Option mask for this reset
		Defaults=0x0001 Reset to defaults
		Swap=0x0004 Swap boot partition

Outputs

_rv	*integer*	Return status (0 = success)

SYNC shutdown(ConstStringZ reason)

Request a system shutdown

privilege level USER

Parameters

Inputs

reason	*string*	String to be placed in the reset log

Outputs

_rv	*integer*	Return status (0 = success)

EVENT rebooting( )

Notification that a system reboot is about to occur

EVENT shuttingDown( )

Notification that a system shutdown is about to occur

SYNC getRunLevel(int* run_level)

Retrieve the current system run level

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
run_level	*integer*	Current run level (0-6)

SYNC setRunLevel(int run_level)

Request the system to transition to a new run level

privilege level ADMIN

Parameters

Inputs

run_level	*integer*	New run level for system (0-6)

Outputs

_rv	*integer*	Return status (0 = success)

EVENT runLevelUpdated( )

Notification that a new system run level has been set

SYNC setSSHServiceEnable(int enable)

Enable or disable SSH access to the system

privilege level ADMIN

Parameters

Inputs

enable	*integer*	Disabled=0	Disable SSH service
		Enabled=1	Enable SSH service

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getSSHServiceEnable(int *enable)

Retrieve the current SSH service enable status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
enable	*integer*	Returned enable status
	Disabled=0	Disable SSH service
	Enabled=1	Enable SSH service

EVENT serviceEnableSSHChanged(int enable)

Issued when the SSH service enable is changed

Parameters

enable	*integer*	
	Disabled=0	Disable interface
	Enabled=1	Enable interface

SYNC setHTTPServiceEnable(int enable)

enable or disable HTTP/SOAP access to the system

privilege level ADMIN

Parameters

Inputs

enable	*integer*	
	Disabled=0	Disable interface
	Enabled=1	Enable interface

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getHTTPServiceEnable(int *enable)

Retrieve the current enable status for web services

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
enable	*integer*	Returned enable status
	Disabled=0	Disable interface

Enabled=1 Enable interface

EVENT serviceEnableHTTPChanged(int enable)

Issued when the HTTP service is enabled or disabled

Parameters

enable	*integer*	New HTTP enable status (0 = disabled, 1 = enabled)

ASYNC startTcpdump(ConstStringZ tcpdump_filter)

invoked to start tcpdump and supply command line arguments to tcpdump program. Once started tcpdump records data into data files and rotates them based on size, each new file rotation will cause a tcpdumpDataReady response.

privilege level ADMIN

Parameters

Inputs

tcpdump_filter	*string*	allows user supplied tcpdump port filter, if null or empty string then default filter is used

ASYNC stopTcpdump( )

invoked to stop tcpdump, after tcpdump is stopped expect one tcpdumpDataReady response that provides any data in the not yet rotated tcpdump data file

privilege level ADMIN

SYNC getTcpdumpEnable(int* tcpdump_enabled)

request to request if tcpdump is started or is stoped

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
tcpdump_enabled	*integer*	0 means tcpdump is stopped, non-zero means tcpdump is started

SYNC getTcpdumpFilename(StringZ filename, size_t filename_size)

get the name of the current tcpdump file for use with Data_openFile. An empty value indicates that no tcpdump file exists.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0 = success -EINVAL = filename is null or filename_size is 0 or less
filename	*string*	a buffer to hold the returned filename

SYNC getTcpdumpFilter(StringZ filter, size_t filter_size)

get the filter of the current tcpdump. If tcpdump is disabled then the last known filter is returned

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status 0 Success -EINVAL Filter is null or filter_size is 0 or less
filter	*string*	a buffer to hold the returned filter

EVENT tcpdumpEnableChanged(int tcpdump_enabled)

event / response sent when tcpdump is started or stoped

Parameters

tcpdump_enabled	*integer*	0 means tcpdump is stopped, non-zero means tcpdump is started

EVENT tcpdumpDataReady(ConstStringZ file_name, int file_size)

Notification sent when a new tcpdump file is available for collection via Data_openFile

Parameters

file_name	*string*	Tcpdump file name
file_size	*integer*	Tcpdump file size in bytes

SYNC setSyslogServer(ConstStringZ hostname)

Configures remote logging. Expect the event syslogServerUpdated if the hostname does change.

privilege level ADMIN

Parameters

Inputs

hostname	*string*	remote host that can send and receive messages to device logging daemon. Use IND name if DNS is operational or dotted quad if not. An empty value will disable remote logging.

Outputs

_rv	*integer*	Return status

0	Success
-EINVAL	Hostname not valid IP address

SYNC getSyslogServer(StringZ hostname, size_t hostname_size)

Returns current remote logging hostname.

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
hostname	*string*	Current configured remote logging hostname, expect IND name, dotted quad, or empty string.

EVENT syslogServerUpdated()

Occurs when remote logging hostname is changed.

SysInfo Class Documentation

Overview

Functions by Group

Other Functions

getInfo

Enumerating keys and values

getInfoEnumerator
enumerateNextInfo

Notifications

infoUpdated
infoDeleted

Setting the log level

setLogLevel
getLogLevel
logLevelChanged

SYNC getInfo(ConstStringZ name, StringZ value, size_t value_size)

Get the system info property value specified by name.

privilege level USER

Parameters

Inputs

name	*string*	the name of the info property to query

Outputs

_rv	*integer*	Return Status
	0	Success
	-EINVAL	Arguments are null
	-ENOENT	Name does not exist
value	*string*	memory buffer to store result

SYNC getInfoEnumerator()

Launches an enumeration process that causes every name, value pair to be reported using the nextInfo response.

privilege level USER

Parameters

Outputs

_rv	*integer*	enumeration handle

SYNC enumerateNextInfo(int handle, StringZ name, size_t name_size, StringZ value, size_t value_size)

Request to an get next item in an info enumeration, name and value contain the current name,value pair in the system info data store

privilege level USER

Parameters

Inputs

handle	*integer*	Enumeration handle from getInfoEnumerator()

Outputs

_rv	*integer*	Return status 0 Success -1 End of enumeration or error
name	*string*	the name of the info property being reported
value	*string*	the value of the info property being reported

EVENT infoUpdated(ConstStringZ name, ConstStringZ value)

Occurs after a system property is created or updated to a new value.

Parameters

name	*string*	the name of the info property being reported
value	*string*	the value of the info property being reported

EVENT infoDeleted(ConstStringZ name)

Occurs after a system property is deleted.

Parameters

name	*string*	the name of the info property being reported

SYNC setLogLevel(int mask)

Set the log mask

privilege level ADMIN

Parameters

Inputs

`mask`	*integer*	New log mask

Outputs

`_rv`	*integer*	Return status (0 = success)

`SYNC getLogLevel(int *mask)`

returns the current log level mask

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	Return status (0 = success)
`mask`	*integer*	returned mask

`EVENT logLevelChanged(int mask)`

issued when the sysinfo log level changes

Parameters

`mask`	*integer*	new log level mask

SysStatus Class Documentation

Overview

Functions by Group

Other Functions

getStatus
printAllKeys
keyStatusChange

Enumerating status values

getKeyEnumerator
enumerateNextKey
enumerateNextKeyValue

Setting the log level

setLogLevel
getLogLevel
logLevelChanged

SYNC getStatus(ConstStringZ key)

Retrieve the status of a key

privilege level USER

Parameters

Inputs

key	*string*	Key to retrieve

Outputs

_rv	*integer*	Status of requested key

SYNC printAllKeys()

Prints all system status entries to the system log

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)

EVENT keyStatusChange(ConstStringZ key, unsigned int value)

Issued when status changes for a key

Parameters

key	*string*	Key that changed
value	*unsigned int*	New value

SYNC getKeyEnumerator()

Start an enumeration of status keys

privilege level USER

Parameters

Outputs

_rv	*integer*	Enumeration handle

SYNC enumerateNextKey(int handle, StringZ key, int key_size, StringZ mkey, int mkey_size)

Get next set of keys in an enumeration

privilege level USER

Parameters

Inputs

handle	*integer*	Enumeration handle from getKeyEnumerator

Outputs

_rv	*integer*	0	Success
		-1	End of enumeration or error
key	*string*	Status key	
mkey	*string*	Dependent master key	

SYNC enumerateNextKeyValue(int handle, StringZ key, int key_size, StringZ mkey, int mkey_size, int *status)

Get next set of keys in an enumeration

privilege level USER

Parameters

Inputs

handle	*integer*	Enumeration handle from getKeyEnumerator

Outputs

_rv	*integer*	0	Success
		-1	End of enumeration or error
key	*string*	Status key	
mkey	*string*	Dependent master key	
status	*integer*	Status of the key	

SYNC setLogLevel(int mask)

Set the log level for the system status daemon

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

returns the current log level mask of system status

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
mask	*integer*	Returned mask

EVENT logLevelChanged(int mask)

issued when the system status log level changes

Parameters

mask	*integer*	New log level mask

TTYMan Class Documentation

Overview

Functions by Group

Other Functions

setPort
getPort

SYNC setPort(lsdevhandle_t dev, ePortSpeed_t speed, ePortFlow_t flow, ePortErase_t erase, ePortShell_t shell)

Set device parameters

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Device ID	
		port0=0x20000000	USB-serial device on USB port 0
		port1=0x20000001	USB-serial device on USB port 1
speed	*unsigned int*	Baud rate	
		1200=0	1200
		2400=1	2400
		4800=2	4800
		9600=3	9600
		19200=4	19200
		38400=5	38400
		57600=6	57600
		115200=7	115200
flow	*unsigned int*	Flow control	
		None=0	None
		HW=1	HW
		SW=2	SW
erase	*unsigned*	Erase character	

	int	Backspace=0	Backspace
		Delete=1	Delete
shell	*unsigned int*	Shell	
		None=0	None
		Passthrough=1	Passthrough
		Sulogin=2	Sulogin
		Admin=3	Admin
		Visca=4	Visca

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getPort(lsdevhandle_t dev, ePortSpeed_t *speed, ePortFlow_t *flow, ePortErase_t *erase, ePortShell_t *shell)

Get device parameters

privilege level ADMIN

Parameters

Inputs

dev	*unsigned int*	Device ID

Outputs

_rv	*integer*	Return status (0 = success)	
speed	*unsigned int*	Baud rate	
		1200=0	1200
		2400=1	2400
		4800=2	4800
		9600=3	9600
		19200=4	19200
		38400=5	38400
		57600=6	57600
		115200=7	115200
flow	*unsigned int*	Flow control	
		None=0	None
		HW=1	HW
		SW=2	SW
erase	*unsigned int*	Erase character	
		Backspace=0	Backspace
		Delete=1	Delete
shell	*unsigned int*	Shell	
		None=0	None
		Passthrough=1	Passthrough
		Sulogin=2	Sulogin

Admin=3	Admin
Visca=4	Visca

Temp Class Documentation

Overview

The Temp daemon provides access to system temperature readings.

Functions by Group

Reading System Temperature

SYNC getNumberOfSensors(int* min, int* max, int* count)

Get the sensor numbering scheme for the device

privilege level USER

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
min	*integer*	minimum sensor number this interface accepts
max	*integer*	maximum sensor number this interface accepts
count	*integer*	total count of sensors this interface manages

SYNC getInfo(size_t sensor, int* current, StringZ status, size_t status_size, StringZ descrip, size_t descrip_size)

Get all fan info for specified fan number

privilege level USER

Parameters

Inputs

sensor	*unsigned int*	the sensor to query

Outputs

_rv	*integer*	Return status

0	Success	
-EINVAL	Current is NULL, status is NULL, or status has 0 size, descrip is NULL, or descrip has 0 size	
-ENODEV	Sensor number specified is not valid	

`current` *integer* current tachometer reading in revolutions per minute

`status` *string* Temperature status

shutdown	Hottest sensor temperature is equal or above shutdown value
overheated	Hottest sensor temperature is equal or above hot value
warning	Hottest sensor temperature is equal or above warm value
normal	Hottest sensor temperature is below warm value
unknown	The system temperature status has not been decided

`descrip` *string* Sensor description

audioDSP	Sensor inside audio DSP on system board
CPU	Sensor inside CPU on system board
clock	Sensor near clock buffer on system board
battery	Sensor near batteries on system board
netPHY	Sensor near network PHY on system board
videoDSP	Sensor inside video board DSP
videoFPGA	Sensor inside video board FPGA
videoCODEC	Sensor inside video board CODEC
videoPCI	Sensor inside video board PCI bridge

`SYNC getSensorTemp(size_t sensor)`

Get specified sensor temperature reading in degrees Celsius

privilege level USER

Parameters

Inputs

sensor	*unsigned int*	the sensor number to investigate see Temp_getNumberOfSensors for allowed range

Outputs

_rv	*integer*	Sensor Temperature or error status
	-1	Bad sensor specified as param
	otherwise	Sensor temperature in degrees Celsius

SYNC getTempStatus(StringZ status, size_t status_size)

Get overall system temperature status

privilege level USER

Parameters

Outputs

_rv	*integer*	length of string copied into buffer or -1 if the input buffer has 0 size
status	*string*	Temperature status
	shutdown	Hottest sensor temperature is equal or above shutdown value
	overheated	Hottest sensor temperature is equal or above hot value
	warning	Hottest sensor temperature is equal or above warm value
	normal	Hottest sensor temperature is below warm value
	unknown	The system temperature status has not been decided

EVENT hottestSensorChange(size_t sensor, int temp, int rangeLow, int rangeHigh, ConstStringZ status)

Indicates when the hottest sensor has changed in sensor number or status

Parameters

sensor	*unsigned int*	the sensor number see Temp_getNumberOfSensors
temp	*integer*	current sensor temperature in degrees Celsius
rangeLow	*integer*	low temperature in degrees Celsius for current temperature range indicated by status param
rangeHigh	*integer*	high temperature in degrees Celsius for current temperature range indicated by status param
status	*string*	Temperature status
	shutdown	Hottest sensor temperature is

	equal or above shutdown value
overheated	Hottest sensor temperature is equal or above hot value
warning	Hottest sensor temperature is equal or above warm value
normal	Hottest sensor temperature is below warm value
unknown	The system temperature status has not been decided

Timer Class Documentation

Overview

Functions by Group

Watchdog Timers

watchdogWarning
watchdogFailure

System Logging

setLogLevel
getLogLevel
logLevelChanged

EVENT watchdogWarning(int pid, int timerid, StringZ info)

Issued when a watchdog timer is about to expire

Parameters

pid	*integer*	PID of process that owns the timer
timerid	*integer*	Timer ID of expiring timer
info	*string*	Timer info string

EVENT watchdogFailure(int pid, int timerid, StringZ info)

Issued when a watchdog has expired

Parameters

pid	*integer*	PID of process that owns the timer
timerid	*integer*	Timer ID of expiring timer
info	*string*	Timer info string

SYNC setLogLevel(int mask)

Set the log level for the pref service

privilege level ADMIN

Parameters

Inputs

mask	*integer*	New log mask

Outputs

_rv	*integer*	Return status (0 = success)

SYNC getLogLevel(int *mask)

returns the current log level mask of dataman

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	Return status (0 = success)
mask	*integer*	returned mask

EVENT logLevelChanged(int mask)

issued when the data manager log level changes

Parameters

mask	*integer*	new log level mask

USBHotplug Class Documentation

Overview

Functions by Group

Other Functions

connectedChanged

EVENT connectedChanged(lsdevhandle_t dev, int connected)

Event indicating device connected state changed

Parameters

dev	*unsigned int*	Device ID	
		0x20000000	USB-serial device on USB port 0
		0x20000001	USB-serial device on USB port 1
connected	*integer*	Connected state	
		0	Disconnected
		1	Connected

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

VDEC Class Documentation

Overview

Functions by Group

Other Functions

ASYNC Open(int transID, int confID, int callID, video_codec_t codec)

Open a channel to a video decoder instance. Allocates the decoder via the VRM

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- Transaction ID for this asynchronous call
confID	*integer*	- Conference associated with this decoder instance
callID	*integer*	- Call associated with this decoder instance
codec	*unsigned int*	- Type of decoder instance to open (e.g., H.264, H.263, etc.)

RESPONSE OpenDone(int transID, int devID)

Response for the two decoder open requests

Parameters

transID	*integer*	- Transaction ID associated with the original request
devID	*integer*	- Handle to the decoder instance

ASYNC Close(int transID, int devID)

Close a previously allocated decoder channel instance

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- Transaction id for this asynchronous call
devID	*integer*	- Handle to the decoder instance to close

RESPONSE CloseDone(int transID, int result)

Response for the Close request

Parameters

transID	*integer*	- Transaction ID for the corresponding close request
result	*integer*	- Result status for the close request, generally 0 indicating success

ASYNC GetResolution(int transID, int devID)

Get the current resolution of the assocated decoder instance

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- Transaction id for this asynchronous call
devID	*integer*	- Handle to the decoder instance to query

RESPONSE GetResolutionDone(int transID, int width, int height)

Response for the GetResolution request

Parameters

transID	*integer*	- Transaction ID for the corresponding get resolution request
width	*integer*	- Width of the currently decoded frame
height	*integer*	- Height of the currently decoder frame

EVENT ResolutionChanged(int devID, int width, int height)

Event indicating that the resolution of the decoder output has changed

Parameters

devID	*integer*	- Decoder instance reporting the change
width	*integer*	- New decoded resolution
height	*integer*	- New decoded height

ASYNC UpdateCodec(int transID, int devID, video_codec_t codec, int confID, int callID)

Update the codec type and information for the assocated decoder instance

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- Transaction id for this asynchronous call
devID	*integer*	- Handle to the decoder instance to query
codec	*unsigned int*	- Type of decoder instance to update to (e.g., H.264, H.263, etc.)
confID	*integer*	- New Conference associated with this decoder instance
callID	*integer*	- New Call associated with this decoder instance

RESPONSE UpdateCodecDone(int transID, int result)

Response for the UpdateCodec request

Parameters

transID	*integer*	- Transaction ID for the corresponding update request
result	*integer*	- Result status for the update request, generally 0 indicating success

ASYNC GetDelay(int transID, int devID)

Get the current decode delay for the assocated decoder instance

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- Transaction id for this asynchronous call
devID	*integer*	- Handle to the decoder instance to query

RESPONSE GetDelayDone(int transID, int delay_ms)

Response for the GetDelay request

Parameters

transID	*integer*	- Transaction ID for the corresponding get delay request
delay_ms	*integer*	- Decode delay for the current codec/frame rate

ASYNC SetDelay(int transID, int devID, int delay_ms)

Set the decoder desired delay for the assocated decoder instance

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- Transaction id for this asynchronous call
devID	*integer*	- Handle to the decoder instance to query
delay_ms	*integer*	- Desired decode delay for the instance

RESPONSE SetDelayDone(int transID, int result)

Response for the SetDelay request

Parameters

transID	*integer*	- Transaction ID for the corresponding set delay request
result	*integer*	- Result status for the set delay request, generally the current actual delay

SYNC setLogLevel(int mask)

Set the Video Input Manager's default log level

privilege level ADMIN

Parameters

Inputs

mask	*integer*	- the new level to use

Outputs

_rv	*integer*	0

SYNC getLogLevel(int *mask)

Get the Video Input Manager's current log level

privilege level ADMIN

Parameters

Outputs

`_rv`	*integer*	0
`mask`	*integer*	- returned log level

`EVENT IntraFrameRequired(int mmcallid, int handle)`

Issued by a decoder when an intra frame is required to continue decoding video

Parameters

`mmcallid`	*integer*	- Call id for which an intra frame is required
`handle`	*integer*	- The id of the decoder that is requesting an I frame

`EVENT InputFramerateChanged(int devID, int fps)`

Notification that the output frame rate for the decoder instance has changed

Parameters

`devID`	*integer*	- Decoder for which the frame rate has changed
`fps`	*integer*	- New frame rate

`EVENT IntraFrameReceived(int mmcallid, int handle)`

Issued by the decoder when an intra frame is received and successfully processed

Parameters

`mmcallid`	*integer*	- Call id for which the intra frame was received
`handle`	*integer*	- Id of the decoder that confirms receipt of I frame

`EVENT logLevelChanged(int mask)`

Notification from the Video Input Manager that the log level has changed

Parameters

`mask`	*integer*	- new log level

VENC Class Documentation

Overview

Functions by Group

Other Functions

ASYNC AddChannel_new(int transID, int confID, int callID, int syncKey, const conf_res_t confRes, const video_codec_t videoCodec)

Add a new encoder channel to the system. This is the currently supported interface

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction ID for response tracking
confID	*integer*	- input conference ID associated with this encoder

callID	*integer*	- input call ID associated with this encoder
syncKey	*integer*	- input magic number to associate with the packet data in the bitstream
confRes	*unsigned int*	- input conference resolution enum
videoCodec	*unsigned int*	- input video codec choice

RESPONSE AddChannelDone(int transID, int handle)

Return path for all forms of AddChannel message

Parameters

transID	*integer*	- output the transID from the corresponding AddChannel command
handle	*integer*	- output the encoder handle for use with future calls, or -1 if the AddChannel failed

ASYNC ConfigureH264Channel(int transID, int handle, const venc_common_caps_t *venc_common_caps, const H264Parameters_t *h264Params)

Configure an encoder for H264 operation

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction id for response tracking
handle	*integer*	- input encoder handle returned from a previous AddChannel command
venc_common_caps	*structure*	- input encoder common parameters
h264Params	*structure*	- input H.264 specific parameters

RESPONSE ConfigureH264ChannelDone(int transID, int status)

Reply to the ConfigureH264Channnel message

Parameters

transID	*integer*	- output transID from the corresponding request
status	*integer*	- output configuration status from the command, 0 on success, a negative number on error

ASYNC ConfigureH263PChannel(int transID, int handle, const venc_common_caps_t *venc_common_caps, const H263PParameters_t *h263P_list_in, int h263P_list_in_count)

Configure an encoder for H263plus operation

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction id for response tracking
handle	*integer*	- input encoder handle returned from a previous AddChannel command
venc_common_caps	*structure*	- input encoder common parameters
h263P_list_in	*array of structure*	- input H.263+ specific parameters

RESPONSE ConfigureH263PChannelDone(int transID, int status)

Reply to the ConfigureH263PChannnel message

Parameters

transID	*integer*	- output transID from the corresponding request
status	*integer*	- output configuration status from the command, 0 on success, a negative number on error

ASYNC ConfigureH263Channel(int transID, int handle, const venc_common_caps_t *venc_common_caps, const H263Parameters_t *h263_list_in, int h263_list_in_count)

Configure an encoder for H263 operation

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction id for response tracking
handle	*integer*	- input encoder handle returned from a previous AddChannel command
venc_common_caps	*structure*	- input encoder common parameters
h263_list_in	*array of structure*	- input H.263 specific parameters

RESPONSE ConfigureH263ChannelDone(int transID, int status)

Reply to the ConfigureH263Channnel message

Parameters

transID	*integer*	- output transID from the corresponding request
status		

	integer	- output configuration status from the command, 0 on success, a negative number on error

ASYNC ConfigureH261Channel(int transID, int handle, const venc_common_caps_t *venc_common_caps, const H261Parameters_t *h261_list_in, int h261_list_in_count)

Configure an encoder for H261 operation

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction id for response tracking
handle	*integer*	- input encoder handle returned from a previous AddChannel command
venc_common_caps	*structure*	- input encoder common parameters
h261_list_in	*array of structure*	- input H.261 specific parameters

RESPONSE ConfigureH261ChannelDone(int transID, int status)

Reply to the ConfigureH261Channnel message

Parameters

transID	*integer*	- output transID from the corresponding request
status	*integer*	- output configuration status from the command, 0 on success, a negative number on error

ASYNC UpdateChannel(int transID, int handle, int confID, int callID, int syncKey, const conf_res_t confRes, const video_codec_t videoCodec)

Update the properties of an encoder channel. Used to Close/Add an encoder without losing the encoder in a race

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction ID for response tracking
handle	*integer*	- input encoder handle to update
confID	*integer*	- input conference ID associated with this encoder
callID	*integer*	- input call ID associated with this encoder
syncKey	*integer*	- input magic number to associate with the packet data in the bitstream

confRes	*unsigned int*	- input conference resolution enum
videoCodec	*unsigned int*	- input video codec choice

RESPONSE UpdateChannelDone(int transID, int handle)

Return path for the UpdateChannel message

Parameters

transID	*integer*	- output the transID from the corresponding AddChannel command
handle	*integer*	- output the encoder handle for use with future calls, or -1 if the UpdateChannel failed. If successful, the encoder handle will be the same one passed in

ASYNC Close(int transID, int handle)

Close the encoder channel stopping the encoder and releasing its resources

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction ID for response tracking
handle	*integer*	- input encoder handle to close

RESPONSE CloseDone(int transID, int result)

Return path for the Close message

Parameters

transID	*integer*	- output transID from the corresponding request
result	*integer*	- output return value from command, 0 on success, a negative number on error

ASYNC GenerateIntraFrame(int transID, int handle)

Force an intra frame to be generated on the specified encoder

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction ID for response tracking
handle	*integer*	- input encoder handle force an iframe on

RESPONSE GenerateIntraFrameDone(int transID, int result)

Response to GenerateIntraFrame command

Parameters

transID	*integer*	- output transID from the corresponding request
result	*integer*	- output return value from command, 0 on success, a negative number on error

ASYNC GetFrameRate(int transID, int handle)

Request the current encoder target frame rate

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction ID for response tracking
handle	*integer*	- input encoder handle to get the frame rate of

RESPONSE GetFrameRateDone(int transID, int frameRate)

Response to GetFrameRate command

Parameters

transID	*integer*	- output transID from the corresponding request
frameRate	*integer*	- output target frame rate of the encoder

ASYNC GetResolution(int transID, int handle)

Request the current encoding resolution from an encoder

privilege level ADMIN

Parameters

Inputs

transID	*integer*	- input transaction ID for response tracking
handle	*integer*	- input encoder handle to get the resolution of

RESPONSE GetResolutionDone(int transID, int width, int height)

Reply to the GetResolution command

Parameters

transID	*integer*	- output transID from the corresponding request
width	*integer*	- output width of the encoded video

height *integer* - output height of the encoded video

ASYNC GetDelay(int transID, int handle)

Get the lip sync delay in the video encode pipeline - the delay incurred based on the current encode parameters

privilege level ADMIN

Parameters

Inputs

transID *integer* - input transaction ID for response tracking

handle *integer* - input encoder handle to get the delay for

RESPONSE GetDelayDone(int transID, int delay_ms)

Reply to the GetDelay command

Parameters

transID *integer* - output transID from the corresponding request

delay_ms *integer* - output milliseconds of delay in the video path, not including additional delays from SetDelay, -1 on error

ASYNC SetDelay(int transID, int handle, int delay_ms)

Set the additional lip sync delay in the video encode pipeline

privilege level ADMIN

Parameters

Inputs

transID *integer* - input transaction ID for response tracking

handle *integer* - input encoder handle to set the delay on

delay_ms *integer* - number of milliseconds to delay vido

RESPONSE SetDelayDone(int transID, int result)

Response to the SetDelay command

Parameters

transID *integer* - output transID from the corresponding request

result *integer* - output return value from command, 0 on success, a negative number on error

SYNC inUse(int handle)

Determine if a specific encoder is already in use

privilege level ADMIN

Parameters

Inputs

handle	*integer*	- input encoder handle to check

Outputs

_rv	*integer*	0 if idle, 1 if in use

SYNC setLogLevel(int mask)

Set the vencManager default log level

privilege level ADMIN

Parameters

Inputs

mask	*integer*	- input log events to output

Outputs

_rv	*integer*	0

SYNC getLogLevel(int *mask)

Get the vencManager's current log level

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0
mask	*integer*	- output log event level

EVENT logLevelChanged(int mask)

Notification that the vencManager log level has changed

Parameters

mask	*integer*	- output new log level

EVENT resolutionChanged(int handle, int width, int height, int fps)

Notification that one of the encoder's resolution or frame rate has changed

Parameters

handle	*integer*	- handle to the encoder that has been modified

`width`	*integer*	- new encode width
`height`	*integer*	- new encode height
`fps`	*integer*	- new encode frame rate

`SYNC GetVideoBitrate(int handle)`

Get the bit rate computed by video for the indicated call. Useful when comm indicates auto bit rate and video chooses a different value based on the remote caps.

privilege level ADMIN

Parameters

Inputs

`handle`	*integer*	- input encoder handle to get the bitrate for

Outputs

`_rv`	*integer*	the bitrate for this handle, -1 on not active, -2 if not supported, -10 on invalid handle

VIDEO_HW Class Documentation

Overview

Functions by Group

Other Functions

SYNC setLogLevel(int mask)

Set the Video Input Manager's default log level

privilege level ADMIN

Parameters

Inputs

mask	*integer*	- the new level to use

Outputs

_rv	*integer*	0

SYNC getLogLevel(int *mask)

Get the Video Input Manager's current log level

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0
mask	*integer*	- returned log level

EVENT logLevelChanged(int mask)

Notification from the Video Input Manager that the log level has changed

Parameters

mask	*integer*	- new log level

SysInfo
SysStatus
TTYMan
Temp
Timer
USBHotplug
VDEC
VENC
VIDEO_HW
VIDEO_IN
VIDEO_OUT
VRM

VIDEO_IN Class Documentation

Overview

Functions by Group

Other Functions

SYNC forceInputType(lsdevhandle_t inputID, Vin_InType type)

Interface for the UI to tell Vin that the input needs to be forced to a specific type for compatibility reasons

privilege level ADMIN

Parameters

Inputs

inputID	*unsigned int*	Input number to change the type of hdmi0=0x00020000 HDMI input dvi0=0x00100000 DVI-I input
type	*unsigned int*	Type to switch to inTypeAny=0 Default (accepts HDMI, DVI, VGA, Component as applicable, Component not currently supported) inTypeDVI=1 DVI input only (will not indicate HDMI or VGA is supported) inTypeVGA=2 VGA input only (will not indicate DVI or HDMI is supported) inTypeDVIVGA=3 DVI/VGA input only (will not indicate HDMI is supported)

Outputs

_rv	*integer*	0 on success, 1 if the type is not applicable (e.g., vga on an hdmi input), 2 on failure

SYNC getInputType(lsdevhandle_t inputID, Vin_InType * type)

Interface for the UI to get the current input type restriction for a specific input

privilege level ADMIN

Parameters

Inputs

inputID	*unsigned int*	Input number to get the type of hdmi0=0x00020000 HDMI input dvi0=0x00100000 DVI-I input

Outputs

_rv	*integer*	0 on success, 1 if something went wrong
type	*unsigned int*	Returned type of the input inTypeAny=0 Default (accepts HDMI, DVI, VGA, Component as applicable, Component not currently supported) inTypeDVI=1 DVI input only inTypeVGA=2 VGA input only inTypeDVIVGA=3 DVI/VGA input only

`SYNC getInputState(lsdevhandle_t inputID, Vin_Info * info)`

Read the current input state as if an inputChanged event had been delivered

privilege level ADMIN

Parameters

Inputs

inputID	*unsigned int*	Input for which the status is desired hdmi0=0x00020000 HDMI input dvi0=0x00100000 DVI-I input

Outputs

_rv	*integer*	0 on success, 1 if something went wrong
info	*structure*	Information about the current input state activeWidth Viewable number of pixels in a line activeHeight Viewable number of lines in a frame totalWidth Total number of pixels in a line (optionally specified) totalHeight Total number of lines in a frame (optionally specified) fieldFreq Number of fields/frames avilable per second signalType Type of signal being received inStateNoSignal=0 No signal inStateHDMI=1 HDMI inStateDVI=2 DVI inStateVGA=3 VGA inStateComponent=4 Component (not supported at this time)

	inStateComposite=5	Composite (not supported at this time)
	inStateSVideo=6	SVideo (not supported at this time)
	inStateCameraUpdate=7	Camera update active
aspectRatio	Aspect ratio of the input video, if known	
	inAspectUnknown=0	Unknown
	inAspect16_9=1	16:9
	inAspect4_3=2	4:3
	inAspect5_4=3	5:4
	inAspect16_10=4	16:10
interlaced	1 if interlaced video is being received (not supported), 0 if progressive	
modeName	Descriptive name of the video mode	

`SYNC getSPDState(lsdevhandle_t inputID, int force, SPD_Info * info)`

Read the current SPD state as if an spdReceived event had been delivered

privilege level ADMIN

Parameters

Inputs

`inputID`	*unsigned int*	Input for which the spd contents are desired
		hdmi0=0x00020000 HDMI input
		dvi0=0x00100000 DVI-I input
`force`	*integer*	Force read of SPD state from hardware (prevents code 2 from being returned, may retrieve garbage)

Outputs

`_rv`	*integer*	0 on success, 1 if something went wrong, 2 if there is no valid SPD for this input
`info`	*structure*	Current SPD data received from the source device
		spd SPD data
		spdSize Number of valid bytes of SPD data

`SYNC getCableState(lsdevhandle_t inputID, int * state)`

Read the current assert 5V state as if a cableDetected event had been delivered

privilege level ADMIN

Parameters

Inputs

`inputID`	*unsigned int*	Input for which the cable detect state is desired

hdmi0=0x00020000 HDMI input
dvi0=0x00100000 DVI-I input

Outputs

_rv	*integer*	0 on success, 1 if something went wrong
state	*integer*	Current cable detection state 0 Cable not detected 1 Cable detected

SYNC setLogLevel(int mask)

Set the Video Input Manager's default log level

privilege level ADMIN

Parameters

Inputs

mask	*integer*	- the new level to use

Outputs

_rv	*integer*	0

SYNC getLogLevel(int *mask)

Get the Video Input Manager's current log level

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0
mask	*integer*	- returned log level

EVENT logLevelChanged(int mask)

Notification from the Video Input Manager that the log level has changed

Parameters

mask	*integer*	- new log level

EVENT inputChanged(lsdevhandle_t inputID, Vin_Info * info)

Notification from the Video Input Manager that the state of an input has changed (e.g., connected, resolution, etc.)

Parameters

inputID	*unsigned int*	Input for which the status has changed hdmi0=0x00020000 HDMI input dvi0=0x00100000 DVI-I input
info	*structure*	Information about the new input state activeWidth Viewable number of pixels in a line activeHeight Viewable number of lines in a frame

totalWidth	Total number of pixels in a line (optionally specified)
totalHeight	Total number of lines in a frame (optionally specified)
fieldFreq	Number of fields/frames avilable per second
signalType	Type of signal being received
	inStateNoSignal=0 — No signal
	inStateHDMI=1 — HDMI
	inStateDVI=2 — DVI
	inStateVGA=3 — VGA
	inStateComponent=4 — Component (not supported at this time)
	inStateComposite=5 — Composite (not supported at this time)
	inStateSVideo=6 — SVideo (not supported at this time)
	inStateCameraUpdate=7 — Camera update active
aspectRatio	Aspect ratio of the input video, if known
	inAspectUnknown=0 — Unknown
	inAspect16_9=1 — 16:9
	inAspect4_3=2 — 4:3
	inAspect5_4=3 — 5:4
	inAspect16_10=4 — 16:10
interlaced	1 if interlaced video is being received (not supported), 0 if progressive
modeName	Descriptive name of the video mode

`EVENT spdReceived(lsdevhandle_t inputID, SPD_Info * info)`

Notification from the Video Input Manager that the contents of the SPD data area have changed

Parameters

`inputID`	*unsigned int*	Input for which the spd has changed
		hdmi0=0x00020000 — HDMI input
		dvi0=0x00100000 — DVI-I input
`info`	*structure*	SPD data received from the source device
		spd — SPD data
		spdSize — Number of valid bytes of SPD data

`EVENT cableDetected(lsdevhandle_t inputID, int state)`

Notification that a cable detect event has occurred (either plug in or removal)

Parameters

`inputID`	*unsigned int*	Input for which the cable detect state has changed hdmi0=0x00020000 HDMI input dvi0=0x00100000 DVI-I input
`state`	*integer*	Current cable detect state 1 Cable present 0 Cable absent (or device off)

VIDEO_OUT Class Documentation

Overview

Functions by Group

Other Functions

SYNC forceOutputType(lsdevhandle_t outputID, Vout_OutType type)

Interface for the UI to tell Vout that the output needs to be forced to a specific type because the output device does not support DDC. In this mode, we will assume default video modes of 720p60 and 1080p60 if we cannot read the DDC

privilege level ADMIN

Parameters

Inputs

outputID	*unsigned int*	Output number to change the type of	
		hdmi0=0x02000000	HDMI out
		dvi0=0x02000001	DVI-I out
type	*unsigned int*	Output type to force	
		outTypeAny=0	Default (DVI, HDMI, VGA, Component accepted)
		outTypeDVI=1	DVI only
		outTypeVGA=2	VGA only
		outTypeComponent=3	Component only

(not currently supported)

Outputs

_rv	*integer*	0 on success, 1 if the type is not applicable (e.g., vga on an hdmi output), 2 on failure

SYNC setAllModes(lsdevhandle_t outputID, int enable)

Enable or disable the all modes override. With the override enabled, the DDC read is skipped and we assume all 3 standard modes are supported (720p60, 1080p60 and 1080p30).

privilege level ADMIN

Parameters

Inputs

outputID	*unsigned int*	Output to modify hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out
enable	*integer*	Enable or disable the override 0 disable 1 enable

Outputs

_rv	*integer*	0 on success, 1 if the outputID is invalid

SYNC getOutputType(lsdevhandle_t outputID, Vout_OutType * type)

Interface for the UI to get the current output type restriction for a specific output

privilege level ADMIN

Parameters

Inputs

outputID	*unsigned int*	Output number to get the type of hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out

Outputs

_rv	*integer*	0 on success, 1 if something went wrong
type	*unsigned int*	Returned type of the output outTypeAny=0 Default (DVI, HDMI, VGA, Component accepted) outTypeDVI=1 DVI only outTypeVGA=2 VGA only

outTypeComponent=3	Component only (not currently supported)

SYNC getOutputModesAvailable(lsdevhandle_t outputID, Vout_Mode * modes, int * modes_count)

Return the list of available video modes as if a modesAvailable event had been generated

privilege level ADMIN

Parameters

Inputs

outputID	*unsigned int*	Output for which the mode list is desired hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out
modes_count	*unsigned int*	On input, maximum size of the list, on output, the number of modes returned

Outputs

_rv	*integer*	0 on success, 1 if something went wrong
modes	*array of structure*	Current modes available totalWidth: Total number of pixels in a line (optionally specified) totalHeight: Total number of lines in a frame (optionally specified) activeWidth: Viewable number of pixels in a line activeHeight: Viewable number of lines in a frame frameRate: Number of frames avilable per second interlaced: 1 if video is interlaced, 0 if progressive isDefault: 1 if information is from a built-in default, 0 if from display information modeName: Descriptive name of the video mode

SYNC getOutputModesIterator(lsdevhandle_t outputID)

Generate an iterator for the list of available video modes. There are a limited number of such iterators available. They expire after 20 seconds of non-use or at the end of the list.

privilege level ADMIN

Parameters

Inputs

outputID	*unsigned int*	Output for which the mode list iterator is desired hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out

Outputs

_rv	*integer*	-1 on failure, a positive integer on success

SYNC getOutputModesByIteration(int iterator, Vout_Mode * mode)

Get a single mode definition based on the current iterator position.

privilege level ADMIN

Parameters

Inputs

iterator	*integer*	Reference for the desired mode

Outputs

_rv	*integer*	0 on success, -1 if there are no more modes available, -2 if the iterator is invalid
mode	*structure*	Returned mode information totalWidth — Total number of pixels in a line (optionally specified) totalHeight — Total number of lines in a frame (optionally specified) activeWidth — Viewable number of pixels in a line activeHeight — Viewable number of lines in a frame frameRate — Number of frames avilable per second interlaced — 1 if video is interlaced, 0 if progressive isDefault — 1 if information is from a built-in default, 0 if from display information modeName — Descriptive name of the video mode

SYNC setOutputMode(lsdevhandle_t outputID, const Vout_Mode * mode)

Set the output device to the specified video mode configuration

privilege level ADMIN

Parameters

Inputs

`outputID`	*unsigned int*	Output for which the mode setting is desired hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out
`mode`	*structure*	Mode to set it to - will be checked against the list of valid modes for the device

totalWidth	Total number of pixels in a line (ignored)
totalHeight	Total number of lines in a frame (ignored)
activeWidth	Viewable number of pixels in a line
activeHeight	Viewable number of lines in a frame
frameRate	Number of frames avilable per second
interlaced	1 if video is interlaced, 0 if progressive
isDefault	1 if information is from a built-in default, 0 if from display information (ignored)
modeName	Descriptive name of the video mode (ignored)

Outputs

`_rv`	*integer*	0 on success, 2 if the mode is invalid for the attached device, 1 for other errors

`SYNC getOutputMode(lsdevhandle_t outputID, Vout_Mode * mode)`

Get the current mode of the output device

privilege level ADMIN

Parameters

Inputs

`outputID`	*unsigned int*	Output to query hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out

Outputs

`_rv`	*integer*	0 on success, 1 for invalid device
`mode`	*structure*	Mode data for the output mode, 0x0 if the output device is not enabled/on

totalWidth	Total number of pixels in a line (optionally specified)
totalHeight	Total number of lines in a frame (optionally specified)
activeWidth	Viewable number of pixels in a line
activeHeight	Viewable number of lines in a frame
frameRate	Number of frames avilable per second
interlaced	1 if video is interlaced, 0 if progressive
isDefault	1 if information is from a built-in default, 0 if from display information
modeName	Descriptive name of the video mode

SYNC getMonitorConnected(lsdevhandle_t outputID, int * connected)

Get the current cable plug/unplug status of the corresponding port

privilege level ADMIN

Parameters

Inputs

outputID	*unsigned int*	Output for which the cable state is desired hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out

Outputs

_rv	*integer*	0 on success, 1 for invalid device
connected	*integer*	Indicates the state of the cable 1 connected 0 disconnected

SYNC getCECPhysicalAddress()

Get the curent CEC Physical Address as reported by the attached primary (HDMI) display

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	The CEC Physcial Address as an unsigned short, 0

indicates no device attached or CEC is unsupported

SYNC setLogLevel(int mask)

Set the Video Output Manager's default log level

privilege level ADMIN

Parameters

Inputs

mask	*integer*	- the new level to use

Outputs

_rv	*integer*	0

SYNC getLogLevel(int *mask)

Get the Video Output Manager's current log level

privilege level ADMIN

Parameters

Outputs

_rv	*integer*	0
mask	*integer*	- returned log level

EVENT logLevelChanged(int mask)

Notification from the Video Output Manager that the log level has changed

Parameters

mask	*integer*	- new log level

EVENT monitorConnected(lsdevhandle_t outputID, int connected)

Notification from the Video Output Manager that a cable plug/unplug event has been detected

Parameters

outputID	*unsigned int*	Output for which the cable state has changed hdmi0=0x02000000 HDMI out dvi0=0x02000001 DVI-I out
connected	*integer*	Indicates the state of the cable 1 connected 0 disconnected

EVENT modesAvailable(lsdevhandle_t outputID, Vout_Mode * modes, int modes_count)

Notification from the Video Output Manager that the list of possible output modes is now available

Parameters

outputID — *unsigned int* — Output for which the modes are now known

hdmi0=0x02000000	HDMI out
dvi0=0x02000001	DVI-I out

modes — *array of structure* — Mode list for the attached display device

totalWidth	Total number of pixels in a line (optionally specified)
totalHeight	Total number of lines in a frame (optionally specified)
activeWidth	Viewable number of pixels in a line
activeHeight	Viewable number of lines in a frame
frameRate	Number of frames avilable per second
interlaced	1 if video is interlaced, 0 if progressive
isDefault	1 if information is from a built-in default, 0 if from display information
modeName	Descriptive name of the video mode

EVENT outputModeChanged(lsdevhandle_t outputID, Vout_Mode * mode)

Reports when the output mode has changed

Parameters

outputID — *unsigned int* — The output for which the mode change has occurred

hdmi0=0x02000000	HDMI out
dvi0=0x02000001	DVI-I out

mode — *structure* — The new mode parameters

totalWidth	Total number of pixels in a line (optionally specified)
totalHeight	Total number of lines in a frame (optionally specified)
activeWidth	Viewable number of pixels in

	a line
activeHeight	Viewable number of lines in a frame
frameRate	Number of frames avilable per second
interlaced	1 if video is interlaced, 0 if progressive
isDefault	1 if information is from a built-in default, 0 if from display information
modeName	Descriptive name of the video mode

`EVENT cecPhysicalAddressChanged(unsigned short newPA)`

Reports when the CEC physical address has changed as a result of a monitor plug event

Parameters

`newPA`	*unsigned int*	The new physical address value, one digit per nybble. 0 indicates no device connected or CEC is unsupported

VRM Class Documentation

Overview

set the compositor background plane to an rgb image.

Functions by Group

Other Functions

StyleTemplateCreate
StyleTemplateDestroy
StyleDisplay
StyleDisplayDone
InputActive
InputActiveDone
InputPause
InputPauseDone
InputResolutionChange
InputResolutionChangeDone
RealDecoderIdAlloc
RealDecoderIdFree
RealDecoderIdReserve
CompositorEnable
CompositorEnableDone
CompositorScaledResolutionChange
CompositorScaledResolutionChangeStart
CompositorScaledResolutionChangeDone
RealEncoderIdAlloc
RealEncoderIdFree
RealEncoderIdReserve
CompositorOptimalResolution
CompositorOptimalResolutionChanged
ReadInputToRgbFile
ReadInputToRgbFileDone
ReadInputToJpgFile
ReadInputToJpgFileDone
ReadInputToYUVFile
ReadInputToYUVFileDone
SetBackgroundColor
SetBackgroundColorDone
SetBackgroundToRgbImage
SetBackgroundToRgbImageDone

SYNC StyleTemplateCreate(int w,int h, vrmBackground background,const vrmWindow *windowList,int windowList_count, int *outStyleId)

Create a style template to be used with compositors.

privilege level ADMIN

Parameters

Inputs

w	*integer*	- input Ignored. Legacy Helium stuff. Ignored in

		helium too.
h	*integer*	- input Ignored. Legacy Helium stuff. Ignored in helium too..
background	*unsigned int*	- inputI Ignored. Legacy Helium stuff. (TODO: not ignored anymore -- Tyler)
windowList	*array of structure*	- input List of windows in the style. See videoRouteManager.h for struc definition.

Outputs

_rv	*integer*	0 on success -1 on error
outStyleId	*integer*	- output Styleid used to identify a style

SYNC StyleTemplateDestroy(int styleId)

Deletes a style template.

privilege level ADMIN

Parameters

Inputs

styleId	*integer*	- input Styleid used to identify a style

Outputs

_rv	*integer*	0 on success -1 on error

ASYNC StyleDisplay(int transId,int compositorId,int styleId,vrmBackground background,vrmAlignment alignment,const int *inputList,int inputList_count)

Applies a style to a particular compositor and provides the input.

privilege level ADMIN

Parameters

Inputs

transId	*integer*	- input transaction ID for response tracking
compositorId	*integer*	- input CompositorId used to identify a compositor. Definited in lsdevices.h
styleId	*integer*	- input Styleid used to identify a style
background	*unsigned int*	- inputI Ignored. Legacy Helium stuff.
alignment	*unsigned int*	- inputI Ignored.
inputList	*array of integer*	- input List of inputs to be used for this compositor with this style. Definited in lsdevices.h.

RESPONSE StyleDisplayDone(int transId)

Return path for the StyleDisplay messages above

Parameters

transId	*integer*	- output transaction ID from the corresponding StyleDisplay command

ASYNC InputActive(int transId,int input,int active)

Makes an input active. If the input is already connected to a compositor it will start sending frames.

privilege level ADMIN

Parameters

Inputs

transId	*integer*	- input Transaction ID for response tracking
input	*integer*	- input id of input defined in lsdevices.h
active	*integer*	- input Sets whether the input is active(1) or not(0).

RESPONSE InputActiveDone(int transId)

Return path for the InputActive messages above

Parameters

transId	*integer*	- output transaction ID from the corresponding InputActive command

ASYNC InputPause(int transId,int input,int active)

Same as inputActive. Function exists to be backward compatible with Helium.

privilege level ADMIN

Parameters

Inputs

transId	*integer*	- input Transaction ID for response tracking
input	*integer*	- input id of input defined in lsdevices.h
active	*integer*	- input Sets whether the input is active or not.

RESPONSE InputPauseDone(int transId)

Return path for the InputPause messages above

Parameters

transId	*integer*	- output transaction ID from the corresponding InputPause command

ASYNC InputResolutionChange(int transId,int input,int w,int h,int fps)

Changes the input resolution. CURRENTLY NOT SUPPORTED.

privilege level ADMIN

Parameters

Inputs

transId	*integer*	- input Transaction ID for response tracking
input	*integer*	- input id of input defined in lsdevices.h
w	*integer*	- input New width of the input.
h	*integer*	- input New height of the input.
fps	*integer*	- input New frames per sec.

EVENT InputResolutionChangeDone(int input)

Event to notify that the input resolution has changed.

Parameters

input	*integer*	- output id of input defined in lsdevices.h

SYNC RealDecoderIdAlloc(int groupBinding, int *outRealDecoderId)

Allocates the next available decoder.

privilege level ADMIN

Parameters

Inputs

groupBinding	*integer*	- input Ignored. Legacy Helium stuff.

Outputs

_rv	*integer*	0 on success -1 on error
outRealDecoderId	*integer*	- output decoderId used to identify the decoder.

SYNC RealDecoderIdFree(int realDecoderId)

Frees a decoder.

privilege level ADMIN

Parameters

Inputs

realDecoderId	*integer*	- input decoderId used to identify the decoder.

Outputs

_rv	*integer*	0 on success -1 on error

SYNC RealDecoderIdReserve(int realDecoderId)

Reserves the decoder requested if it not already in use. .

privilege level ADMIN

Parameters

Inputs

realDecoderId	*integer*	- input decoderId used to identify the decoder.

Outputs

_rv	*integer*	0 on success -1 on error

ASYNC CompositorEnable(int transId,int compositorId,int enable)

Enables/Disables a compositor. Enabling lets styles be applied to the compositor. Disabling stops the compositor.

privilege level ADMIN

Parameters

Inputs

transId	*integer*	- input Transaction ID for response tracking
compositorId	*integer*	- input CompositorId used to identify a compositor. Definited in lsdevices.h
enable	*integer*	- enable(1) or disable(0) the compositor

RESPONSE CompositorEnableDone(int transId)

Return path for the CompositorEnable messages above

Parameters

transId	*integer*	- output transaction ID from the corresponding CompositorEnable command

ASYNC CompositorScaledResolutionChange(int transId,int compositorId, int w,int h, int fps)

Changes the compositors output/scaled resolution. CURRENTLY NOT SUPPORTED.

privilege level ADMIN

Parameters

Inputs

transId	*integer*	- input Transaction ID for response tracking
compositorId	*integer*	- input CompositorId used to identify a compositor. Definited in lsdevices.h
w	*integer*	- input Width of output of the compositor.
h	*integer*	- input Height of output of the compositor.
fps	*integer*	- input Frames per sec of output of the compositor. .

EVENT CompositorScaledResolutionChangeStart(int compositorId,int oldW,int oldH, int newW, int newH, int oldFps, int newFps)

Return path for the CompositorEnable messages above

Parameters

compositorId	*integer*	- output compositorId for compositor changing resolution.
oldW	*integer*	- output Old widht.
oldH	*integer*	- output Old height.
newW	*integer*	- output New widht.
newH	*integer*	- output New height.
oldFps	*integer*	- output Old fps.
newFps	*integer*	- output New fps.

EVENT CompositorScaledResolutionChangeDone(int compositorId)

Return path for the CompositorEnable messages above

Parameters

compositorId	*integer*	- output compositorId from the corresponding CompositorEnable command

SYNC RealEncoderIdAlloc(int groupBinding, int *realEncoderId)

Allocates the next available encoder.

privilege level ADMIN

Parameters

Inputs

groupBinding	*integer*	- input Ignored. Legacy Helium stuff.

Outputs

_rv	*integer*	0 on success -1 on error
realEncoderId	*integer*	- output encoderId used to identify the encoder.

SYNC RealEncoderIdFree(int realEncoderId)

Frees a encoder.

privilege level ADMIN

Parameters

Inputs

realEncoderId	*integer*	- input encoderId used to identify the encoder.

Outputs

_rv	*integer*	0 on success -1 on error

SYNC RealEncoderIdReserve(int realEncoderId)

Reserves the encoder requested if it not already in use. .

privilege level ADMIN

Parameters

Inputs

realEncoderId	*integer*	- input encoderId used to identify the encoder.

Outputs

_rv	*integer*	0 on success -1 on error

SYNC CompositorOptimalResolution(int compositorId, int * w, int * h, int * fps, int * type)

Returns the optimal configuration for the compositor.

privilege level ADMIN

Parameters

Inputs

compositorId	*integer*	- input compositor to query

Outputs

_rv	*integer*	0 on success, -1 on error
w	*integer*	- output optimal width
h	*integer*	- output optimal height
fps	*integer*	- output optimal frame rate
type	*integer*	- output type of video on the compoistor 0 - primary video - aka camera 1 - secondary video - aka pc 2 - decoded video

3 - multiple sources

EVENT CompositorOptimalResolutionChanged(int compositorId, int w, int h, int fps, int type)

Indicates that the optimal resolution for a compositor has changed, due to an input resolution change

Parameters

compositorId	*integer*	- compositor that has changed
w	*integer*	- optimal width
h	*integer*	- optimal height
fps	*integer*	- optimal fps
type	*integer*	- type of video on the compositor

0 - primary video - aka camera
1 - secondary video - aka pc
2 - decoded video
3 - multiple sources

ASYNC ReadInputToRgbFile(int transId,int input,int w,int h,ConstStringZ filename)

read the most recent buffer from the input, assuming a size of wxh, and write in rgb format to the file

privilege level ADMIN

Parameters

Inputs

transId	*integer*	the transaction ID
input	*integer*	the input id
w	*integer*	the width
h	*integer*	the height
filename	*string*	the filename to write to.

RESPONSE ReadInputToRgbFileDone(int transId)

ReadInputToRgbFile callback

Parameters

transId	*integer*	the transaction ID

ASYNC ReadInputToJpgFile(int transId,int input,int w,int h,int scaled_w,int scaled_h,ConstStringZ filename)

read the most recent buffer from the input, assuming a size of wxh, and write in

jpeg format to the file

privilege level ADMIN

Parameters

Inputs

transId	*integer*	the transaction ID
input	*integer*	the input id
w	*integer*	the width
h	*integer*	the height
scaled_w	*integer*	the desired/target scaled width
scaled_h	*integer*	the desired scaled height
filename	*string*	the filename to write to.

RESPONSE ReadInputToJpgFileDone(int transId)

ReadInputToJpgFile callback

Parameters

transId	*integer*	the transaction ID

ASYNC ReadInputToYUVFile(int transId,int input,int w,int h,ConstStringZ filename)

read the most recent buffer from the input, assuming a size of wxh, and write in jpeg format to the file

privilege level ADMIN

Parameters

Inputs

transId	*integer*	the transaction ID
input	*integer*	the input id
w	*integer*	the width
h	*integer*	the height
filename	*string*	the filename to write to.

RESPONSE ReadInputToYUVFileDone(int transId)

ReadInputToJpgFile callback

Parameters

transId	*integer*	the transaction ID

ASYNC SetBackgroundColor(int transId,int compositorId,int r,int g,int b)

API to set a compositor background plane to a solid rgb color value

privilege level ADMIN

Parameters

Inputs

transId	*integer*	the transaction ID
compositorId	*integer*	compositor id
r	*integer*	red component
g	*integer*	green component
b	*integer*	blue component

RESPONSE SetBackgroundColorDone(int transId)

SetBackgroundColor callback

Parameters

transId	*integer*	the transaction ID

ASYNC SetBackgroundToRgbImage(int transId,int compositorId,ConstStringZ filename)

API set the compositor background plane to the image contained in the file

privilege level ADMIN

Parameters

Inputs

transId	*integer*	the transaction ID
compositorId	*integer*	compositor id
filename	*string*	image file

RESPONSE SetBackgroundToRgbImageDone(int transId)

SetBackgroundToRgbImage callback

Parameters

transId	*integer*	the transaction ID