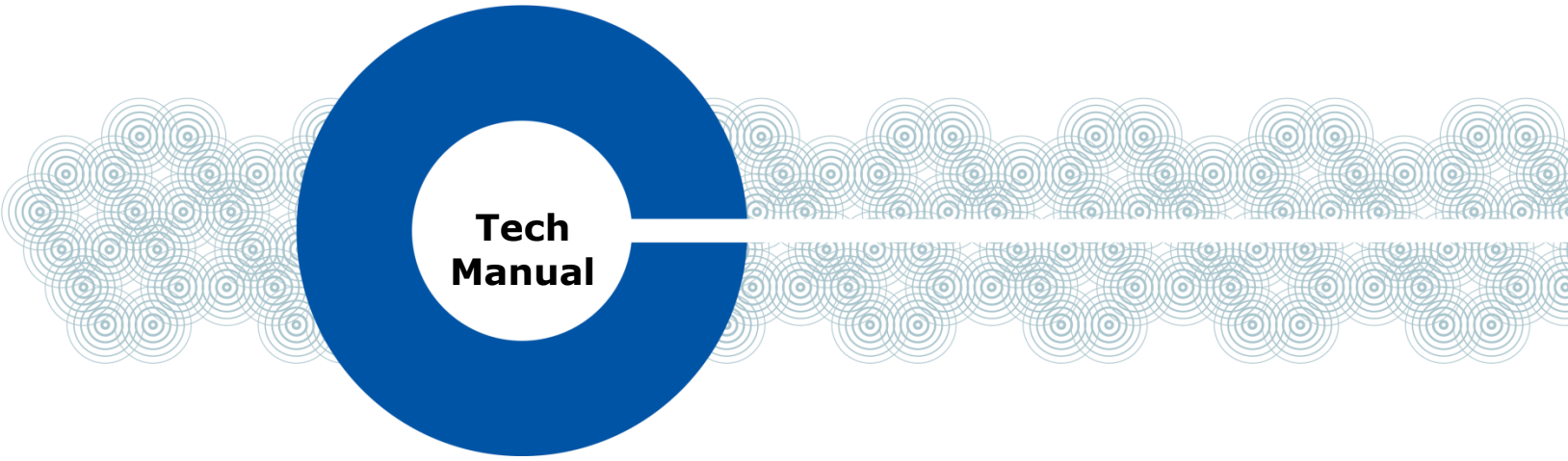


SVoIP Gateway API

Version 1.12

September 2020

Table of Contents

HISTORY & CHANGE LOG

SUBMISSION DATES

Change Log			
Date	**By**	**Details**	**Version**
7/15/19	BD	Draft	1.0
8/9/19	BD	Changed TOC fixed byte order on several commands.	1.1
8/21/19	TA	1. Added RTP Channel Add/Remove command and Status Message for client creation of dynamic RTP channels. This can be used for clients to initiate RTP comms from the client itself or between other gateways/devices. 2. Added Conference Name to conference add command and status message. The clients that didn't create the conference will want to show the name of the conference in their UI. All clients receive conference status message when any client creates a conference.	1.2
9/24/19	WP	1. Added command to enable/disable conference 2. Minor data type edits in JSON section	1.3
10/3/19	TA	1. Added command to send DTMF digits out a channel 2. Minor edits	1.4
11/20/19	BD	Edited response structures for clarity on payload size	1.5
12/2/19	BD	Added ""channel_select_position": 1 to analog port json structure	1.6
12/12/19	TA	1. Added mute rx/tx command 2. Added start/stop ringback command 3. Added "api_provides_ringback" to port config structure	1.7
1/9/2020	BD	Fixed mute and ringback command reference to reflect proper command byte	1.8
1/24/2020	BD	Modified the abort call command description to reflect proper use.	1.9
2/12/2020	BD	Modified to allow SIP and RTP channels to be muted	1.10
2/28/2020	BD	Added channel enable command	1.11
9/10/2020	IO	Added the ability to save RTP channels and conversations to JSON config (Persist)	1.12

I. SVoIP Gateway Description

The SVoIP Gateway provides a a communications link between secure and unsecure air/ground radio assets. Source selection and conferencing between resources is accomplished via a TCP/IP link from a computer client and the SVoIP Gateway. The gateway provides the ability to create dynamic and static conferences between multiple local and remote resources using a web interface and custom application programming interface (API).

In addition to conferencing, the SVoIP Gateway provides a SIP call control mechanism allowing users the ability to access endpoints in a SIP IP PBX environment.

This document is intended to provide detailed technical information on how to utilize the gateway for your application.

Up to seven gateways and as many as six clients can be linked using the API and web interface allowing users the ability to create conferences among various VoIP and analog channels.

II. Port Pinouts

The following diagrams illustrate the proper pin assignments for the relevant port types.

The SVoIP Gateway consists of two individual Clear-Com CG-X4 Gateways linked together via an IP connection. Status and control messages are constantly exchanged between each of the two gateways and voice intercommunication from one device to the other is accomplished by pre-set RTP (Real-time Voice Protocol) unicast links.

Analog VF Ports	
RJ-45 PIN	Description
1	+ 4 Wire Receive
2	— 4 Wire Receive
3	M Lead
4	+ 2 Wire Rcv/Xmt
5	— 2 Wire Rcv/Xmt
6	M Lead Return
7	COR (E Lead)
8	GND (E Lead)

Ethernet Ports	
RJ-45 PIN	Description
1	+ Transmit Data
2	— Transmit Data
3	+ Receive Data
4	- Receive Data
5	N/A
6	N/A
7	N/A
8	N/A

III. Channels

Channels are user-defined RTP, SIP, or analog ports within a gateway.

Analog ports are static channels based on the hardware configuration of the gateway. Typically, these include six 4-wire or 2-wire ports.

RTP and SIP channels are VoIP links to various systems. These channels are setup via the web interface. SIP channels are used to communicate to SIP PBX systems such as Cisco Unified Communications Manager, Avaya, or Asterisk servers. RTP channels are typically used for gateway-to-gateway communication or other third-party VoIP-based systems such as voice logging servers.

A channel, by default, simply provides a means for communication but they need to be added to conferences to link channels from one resource to another.

All channels within each gateway have a unique 32-bit channel ID. All API commands to an individual blade that reference a channel will need to utilize this channel ID. Thus, on startup the client application must download the configuration of each gateway it wants to communicate to (See the "Get JSON Config" command).

Voice is carried between gateways by utilizing standard unicast or multicast RTP streams encoded as G.711 by default. Each stream is a packetized transmission of

voice content sourced between any of the gateway's physical channels and/or TDM bus. Typically, these streams are added into individual "mixers" (called "conferences") that include one or more additional channel resources such as TDM timeslots or actual mic/speaker connections.

IV. Conferences

Once channels are created, they need to be added to conferences before they bridge audio between other resources on the same gateway or between gateways. Static conferences can be created via the web interface or "on-the-fly" via the SVoIP API.

A typical example of establishing a two-way communication link between gateways would be:

1. *Create an RTP multicast channel on both gateway 1 and gateway 2*
2. *Create a conference resource on both gateway 1 and gateway 2*
3. *Add the RTP channel on gateway 1 to the local conference on gateway 1*
4. *Add the RTP channel on gateway 2 to the local conference on gateway 2*
5. *Add the local analog channel 1 to the local conference on gateway 1*
6. *Add the local analog channel 2 to the local conference on gateway 2*
7. *The link is established between channel 1 and 2 between gateways*

V. Terminal Tools

To enter the terminal, connect your Telnet program to the secondary gateway's IP address. You will be presented with a user/password prompt. The default credentials are the same as those configured in the web UI.

Certain terminal commands may prove helpful in troubleshooting and can be viewed by typing "help" in the terminal screen.

A. Viewing Debug Information in Terminal

At times it may be helpful to view real time logs to troubleshoot a problem with the operation of SVoIP gateways. You can view logs by enabling the syslog viewer in the telnet client.

Syntax:
syslog debug[ENTER]
debug on[ENTER]

VI. TCP Control of Gateways

The SVoIP Gateway utilizes a TCP socket architecture for third-party clients to communicate between all gateways individually. Third party clients connect to each gateway to initiate commands. Messages from gateways are distributed to all connected clients with the exception of command responses. Some commands should be directed to individual gateways (such as SIP-related commands) while others can be directed to all gateways (such as the heartbeat command).

VII. API

Each gateway runs a simple socket server to handle remote messaging, status, and RoIP signaling. It is independent of VoIP operation with the exception of SIP call setup and messaging.

Each gateway, by default, listens on TCP port 8676 for messages from a client application. As gateways are added to the system, the client must individually address each which will form a "system array" of gateways. Once the socket is connected your application will begin to receive status and signaling messages.

General Protocol Definition

Gateway/Client ID – this is the client or remote gateway's unique address which is always 6-bytes long. Third-party client applications should use pseudo-random IDs or other unique ID (e.g. MAC address) to define their own application ID.

Message type – the message number.

Header Checksum – this is a header error detection scheme based on the CRC-16-CCITT standard. It uses a polynomial of 0x1021 for calculation.

Payload Size – this is the number of bytes of the payload (all bytes after the checksum bytes).

Payload Checksum – This is a payload error detection scheme based on the CRC-16-CCITT standard. It uses a polynomial of 0x1021 for calculation. If Payload only contains the Payload Checksum the Checksum is set to 0.

VIII. API Commands

A. Authenticate

This command contains authentication information for the remote gateway. It is designed to exchange information about the client to determine if communication is authorized between hosts. Authentication must occur after the socket connection is made or the server will reset the connection. See the section on Authentication later in this document.

Byte 0-5: Client Id
Byte 6-7: Message Type (0x00)
Byte 8-11: Payload Size
Byte 12-13: Header Checksum
Byte 14: Authentication Message Type:

- 0x00 – Request
  - Byte 15-16: Payload Checksum
- 0x01 – Challenge
  - Byte 15: Authentication Mode (0x00 – Default)
  - Byte 16-23: 64 bit Challenge Code
  - Byte 24-25: Payload Checksum
- 0x02 – Answer
  - Byte 15-22: 64 bit Challenge Answer
  - Byte 23-24: Payload Checksum
- 0x03 – Response
  - Byte 15: Status (0x00 Fail, 0x01 Success)
  - Byte 16-17: Payload Checksum

B. Disconnect

This command tells the gateway to disconnect from the socket. This is a clean way to "log off" a remote gateway. If a client apps that crashes is unplugged from the network the remote gateway will detect (see the "Heartbeat" command) that condition and clean up the socket.

Byte 0 - 5: Client Id
Byte 6 - 7: Message Type (0x04)
Byte 8 - 11: Payload Size
Byte 12 – 13: Header Checksum
Byte 14 - 15: Payload Checksum

Returns:
Nothing if successful

C. Heartbeat

This command is used to let another device know it is still operational. The server (typically the gateway itself) will send this message every 5 seconds and it expects the client to send one as well. Clients are responsible for reestablishing the socket connection after the server stops responding. The server will reset the socket connection and clean up a client after 15 seconds of no heartbeat.

Byte 0 - 5: Client Id
Byte 6 - 7: Message Type (0xFFFF)
Byte 8 - 11: Payload Size
Byte 12 - 13: Header Checksum
Byte 14 - 15: Payload Checksum

D. Get JSON Config

This command is used to request the entire configuration structure in JSON format. This is required for clients to be informed of unique IDs for pre-configured channels and ports and allow conferencing commands to be called during runtime. See the section on JSON format later in this document.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x15)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 18 – 19:	Payload Checksum

Returns:

byte 0 – 5:	Gateway Id
byte 6 – 7:	Message Type (0x15)
byte 8 – 11:	Payload Size
byte 12 – 13:	Header CRC
byte 14 – (Payload Size – 2):	JSON structure
Last two bytes:	Payload CRC

E. SIP Call URI

This command is used to initiate a call to a SIP URI.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x20)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18 – (Payload Size – 2):	IP URI to Call
Last Two Bytes:	Payload Checksum

Returns:
See Command Response Message

F. Send DTMF On Channel

This command is used to send DTMF to a channel. Any channel can send DTMF but this may primarily be used on a SIP channel.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x28)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	ChannelID
Byte 18 – (Payload Size – 2):	DTMF digits
Last Two Bytes:	Payload Checksum

Returns:
See Command Response Message

G. Answer/Reject SIP Inbound Call

This command is used to answer an inbound SIP call for a specific account.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x22)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18:	Answer – 1, Reject – 0
Byte 19 –20:	Payload Checksum

Returns:
See Command Response Message

H. Disconnect (hang-up) SIP Call

This command is used to hang-up an active SIP call for a specific account.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x23)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18 - 19:	Payload Checksum

Returns:
See Command Response Message

I. Hold SIP Call

This command is used to place a SIP call on/off hold for a specific account.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x24)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18:	Hold – 1, Off-Hold – 0
Byte 19 - 20:	Payload Checksum

Returns:
See Command Response Message

J. SIP Abort Call

This command is used to abort a call that was initiated from command 0x20. It will cancel outbound call and clean up the session. Use this function to stop a call to an endpoint that is ringing but has not answered.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x26)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18 - 19:	Payload Checksum

Returns:
See Command Response Message

K. PTT On Channel

This command is used to set the radio PTT on a specific channel. For endpoints that require radio keying while in a conference, VOX will be used to key the radio.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x11)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	ChannelID
Byte 18:	Key – 1, De-Key—0
Byte 19 –20:	Payload Checksum

Returns:
See Command Response Message

L. Enable/Disable Conference

This command is sent by clients to enable or disable an existing conference. Useful if pre-defined conferences are setup with overlapping channel members.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x39)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Bytes 14 - 17:	ConferenceID
Bytes 18:	(1 = enable conference, 0 = disable conference)
Bytes 19 - 20:	Payload Checksum

Returns:
Conference Status Message to all clients if successful.
Command response message to initiating client if unsuccessful

---

M. Create/Delete Conference

This command creates a dynamic conference that can be used during runtime. This command does not save newly created conferences between power cycles and won't allow conferences that are statically defined via the web interface to be deleted.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x36)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14:	(1 = Create, 0 = Delete)
Byte 15:	(1 = Persist/Save, 0 = Do not persist)
Bytes 16 – 19:	If byte 14 is 0, ConferenceID of conference to be removed
Bytes 20 – (Payload Size – 2):	If byte 14 is 1, Conference name of the conference being added
Last Two Bytes:	Payload Checksum

Returns:
Conference Status Message to all clients if successful.
Command response message to initiating client if unsuccessful

---

N. Update Conference Member

This command is sent by clients to add or remove a specific channel to/from an existing conference.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x35)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Bytes 14 - 17:	ChannelID to add
Bytes 18 - 21:	ConferenceID
Bytes 22:	(1 = Add channel, 0 = Remove channel)
Bytes 23 - 24:	Payload Checksum

Returns:
Conference Status Message to all clients if successful.
Command response message to initiating client if unsuccessful

---

O. Create RTP Channel

This command is sent by clients to add or remove a specific RTP channel. This will not remove statically defined RTP channels created in the Web UI. Eight RTP channels are available to create dynamically per gateway. Note that when using multicast the peer UDP receiver port will also be the same as the local UDP receiver port so take care not to overlap already allocated UDP receiver ports in your client.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x37)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Bytes 14:	(1 = Create, 0 = Delete)
Byte 15:	(1 = Persist/Save, 0 = Do not persist)
Bytes 16 - 19:	If byte 14 is 0, ChannelID of channel to be removed
Bytes 20:	(0 = Unicast, 1 = Multicast)
Bytes 21 - 23:	UDP Port of Peer Receiver (if unicast)
Bytes 24 – 38:	Peer IP Address (IP address of remote host)
Bytes 39 – (Payload Size – 2):	If byte 14 is 1, Channel name of the channel being added
Last Two Bytes:	Payload Checksum

Returns:
Channel Add/Remove Status Message to all clients if successful.
Command response message to initiating client if unsuccessful

---

P. Set Ringback on Analog Channel

This command is sent by clients to start or stop ringback on a given analog channel.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x40)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	ChannelID
Byte 18:	Start Ringback – 1, Stop Ringback—0
Byte 19 –20:	Payload Checksum

Returns:
See Command Response Message

Q. Set Mute on Analog Channel

This command is sent by clients to set mute options on the TX and/or RX side of a given analog, SIP, or RTP channel.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x41)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	ChannelID
Byte 18:	Mute Receive– 1, Unmute Receive—0
Byte 19:	Mute Transmit– 1, Unmute Transmit—0
Byte 20 –21:	Payload Checksum

Returns:
See Command Response Message

R. Set SIP/RTP Channel Enabled

This command is sent by clients to enable or disable SIP or RTP channels.

Byte 0 - 5:	Clent Id
Byte 6 - 7:	Message Type (0x42)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	ChannelID
Byte 18:	Enabled– 1, Disabled—0
Byte 19 –20:	Payload Checksum

Returns:
See Command Response Message

IX. API Messages from Gateways

A. Heartbeat

This message is used to let another device know it is still operational. The server (typically the gateway itself) will send this message every 5 seconds and it expects the client to send one as well.

Byte 0 - 5:	Gateway Id
Byte 6 - 7:	Message Type (0xFFFF)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 15:	Payload Checksum

B. Gateway Event

Remote gateway hardware event notification (TX, RX, COR, PTT, RTP Session, SIP status).

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x02)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14:	Event Type (0 – COR, 1 – TX, 2 – RX, 3 – PTT, 6 – SIP State)
For Event Type 0 – 3:	
Byte 15 - 18:	Endpoint ChannelID
Byte 19:	State (active – 0x01, inactive 0x00)
Byte 20 - 21:	Payload Checksum
For Event Type 6:	
Byte 15 - 18:	SIP ChannelID
Byte 19:	State (0 – Hold, 1 – Off Hold)
Byte 20-21	Payload Checksum

C. SIP Inbound Call from

This message is used to inform a client/server that SIP URI is calling a specific account. Client should handle this message and play out a ringing sound to client application indicating a call is inbound.

Byte 0 - 5:	Gateway Id
Byte 6 - 7:	Message Type (0x21)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18 – (Payload Size – 2):	SIP URI that is calling
Last Two Bytes:	Payload Checksum

D. SIP Call Answered from URI

This message is used to inform a client/server that SIP URI has answered and connected. This message comes after the SIP endpoint has accepted or connected to a client-answered call or client-initiated call. For inbound calls that are providing client ringing sounds, ringing should stop upon receipt of this message.

Byte 0 - 5:	Client Id
Byte 6 - 7:	Message Type (0x25)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Byte 14 - 17:	SIP ChannelID
Byte 18 – (Payload Size – 2):	SIP URI that has answered/connected
Last Two Bytes:	Payload Checksum

E. SIP Call Rejected / Aborted

This message is sent to clients by the gateway when an inbound or outbound SIP call is aborted or rejected by the distant party.

Byte 0 - 5:	Gateway Id
Byte 6 - 7:	Message Type (0x27)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Bytes 14 - 17:	Sip Channel ID
Bytes 18 - 19:	Payload Checksum

F. Conference Status Message

This message informs all connected clients of success or failure of conference operations. If the operation is to create a conference, the ID of the successfully added conference is returned to all connected clients.

Byte 0 - 5:	Gateway Id
Byte 6 - 7:	Message Type (0x33)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Bytes 14:	Operation (See operations table)
Bytes 15-18:	ConferenceID
Bytes 19-22:	ChannelID
Bytes 23:	Status (See status descriptions)
Bytes 24 - (Payload Size – 2):	Conference Name if Status = 0
Last Two Bytes:	Payload Checksum

Operation Descriptions

Operation Value	Description
0	Create conference
1	Remove conference
2	Add channel to conference
3	Remove channel from conference
4	Enable conference
5	Disable conference

Satus Descriptions

Status Value	Description
0	Operation successful
TBD	TBD

G. RTP Channel Add/Remove Status Message

This message informs all connected clients of success or failure of channel Add/Remove operations. If the operation is to create a channel, the ID of the successfully added channel is returned to all connected clients.

Byte 0 - 5:	Gateway Id
Byte 6 - 7:	Message Type (0x38)
Byte 8 - 11:	Payload Size
Byte 12 - 13:	Header Checksum
Bytes 14:	Operation (See operations table)
Bytes 15-18:	ChannelID
Bytes 19-21:	UDP Receiver port for Client to use for Communication
Bytes 22:	Status (See status descriptions)
Bytes 23 - (Payload Size – 2):	Channel Name if Status = 0
Last Two Bytes:	Payload Checksum

Operation Descriptions

Operation Value	Description
0	Create channel
1	Remove channel

Satus Descriptions

Status Value	Description
0	Operation successful
1	No available RTP Ports
2	Remove Channel Failed
TBD	TBD

H. Command Response

For commands that do not define a specific response/return, this message will be returned to the client. This response is returned only to the client that initiated the command.

Byte 0 - 5:	Gateway Id
Byte 6 - 7:	Message Type (0xFFFE)
Byte 8 - 11:	Payload Size
Byte 12 – 13:	Header Checksum
Byte: 14	Command
Byte: 15	Response Code
Bytes 16 - 17:	Payload Checksum

Response Codes

Response Value	Description
0	Operation successful
1	Operation not permitted
12	Device is out of memory
16	Device or resource is busy
22	DTMF Send error (not in call/inactive channel)
63	Conference or channel ID is invalid
64	Endpoint is not in SIP call or no SIP call available
65	Create conference failed
66	Remove conference failed
67	Add channel to conference failed
68	Remove channel from conference failed
69	No Available RTP Channels
70	Enable/Disable conference failed

X. JSON Format

Clients are required to download gateway configuration from each connected device. This allows clients to pass proper channel or conference IDs into various API commands. The primary JSON information the client will need to capture are channels, conferences, and blade ports.

A. Channels Object

The "channel" object defines SIP and RTP resources defined via the web interface. Clients will be required to send the channelID for most API commands that control channel interfaces.

Channel Types		
Type	Value	Description
RTP	0	An RTP channel is a unicast or multicast VoIP link for communication to remote gateways or third-party equipment such as voice loggers.
SIP	1	SIP channels are used to communicate to SIP PBX systems such as Cisco Unified Communications Manager, Avaya, or Asterisk servers
Analog	2	Analog channels are physical 4-wire or 2-wire ports installed in the gateway.


```
"channels": {
        "channel":
            [
                {
                "channelName": "TestRtp",
                "channelID": 2147483647,
                "channelType": 0,
                "active": true,
                "channelInfo": {
                        "rtpPeer": "10.1.1.245",
                        "rxPort": "64000",
                        "txPort": "64000",
                        "multicastEnabled": "0",
                        "silenceSuppressionTrigger": "2",
                        "codec": "1",
                        "mcastTTL": "",
                        "noiseSuppression": "0",
                        "firewall_keep_alive": "0",
                        "keying": "0"
                        }
                },
                {
                "channelName": "TestSIP",
                "channelID": 1457114332,
                "channelType": 1,
                "active": true,
                "channelInfo": {
                        "username_number": "505",
                        "display_name": "505",
                        "description": "505",
                        "register_domain": "1",
                        "domain_name": "10.1.1.38",
                        "proxy_username": "10.1.1.38",
                        "proxy_password": "505",
                        "proxy_address": "505",
                        "proxy_port": "",
                        "local_port": "5060",
                        "transportType": "0",
                        "connect_on_start": false,
                        "recall": false,
                        "autoAnswer": false,
                        "call_uri": "",
                        "call_uri2": "",
                        "translate_dtmf": false,
```

```
                        "AGC": true,
                        "silence_suppression": true,
                        "JitterCriticalDepth": "300",
                        "JitterTargetDepth": "90",
                        "session_timer": "102",
                        "play_notification": true
                }
        }
    ]
},
```

B. Conferences Object

The "conference" object defines a virtual conference in which channels can be added in order to mix audio between them. Clients will be required to send the conferenceID for most API commands that control interfaces.

```
"conferences": {
        "conference":
           [
                {
                "conferenceName": "Conference 1",
                "conferenceID": 2257483647,
                "active": 1,
                "memberChannels":
                {

                    [
                        {
                        "channelID": 2147483647,
                        "channelType": 0,
                        },
                        {
                        "channelID": 1457114332,
                        "channelType": 1,
                        }
                    ]
                }
            ]
            }
```

C. Blade Object

The "blade" object defines physical analog ports installed in the blade. Clients will be required to send the associated channelID for most API commands that control interfaces.

```
"blade": {
        "name": "blade name",
        "type": 98,
        "modules": [
                {
                "name": "module_1",
                "type": "476-153",
                "configuration": {
                "enabled": true
                },
                "ports": [
                        {
                        "channelID": "-464559877",
                        "channelName": "port_1",
                        "channelType": "2",
                        "configuration": {
                                "channel_select_position": 1
                                "api_provides_ringback": false,
                                "lp_filter_300hz_enabled": false,
                                "hp_filter_300hz_enabled": false,
```

```
                                        "custom_filter_enabled": false,
                                        "ptt_polarity_normal": true,
                                        "cor_polarity_normal": true,
                                        "input_impedance_600": true,
                                        "transmit_level_gain_db": 0,
                                        "receive_level_gain_db": 0,
                                        "audio_delay": {
                                                "receive_ticks": 0,
                                                "transmit_ticks": 1000
                                        },
                                        "transmit_threshold": {
                                                "reporting_enabled": true,
                                                "level_db": -35
                                        },
                                        "receive_threshold": {
                                                "reporting_enabled": true,
                                                "level_db": -35
                                        },
                                "channelID": "-1677694191",
                                "channelName": "port_2",
                                "channelType": "2",
                                "configuration": {
                                        "lp_filter_300hz_enabled": false,
                                        "hp_filter_300hz_enabled": false,
                                        "custom_filter_enabled": false,
                                        "ptt_polarity_normal": true,
                                        "cor_polarity_normal": true,
                                        "input_impedance_600": true,
                                        "transmit_level_gain_db": 0,
                                        "receive_level_gain_db": 0,
                                        "audio_delay": {
                                                "receive_ticks": 0,
                                                "transmit_ticks": 1000
                                        },
                                        "transmit_threshold": {
                                                "reporting_enabled": true,
                                                "level_db": -35
                                        },
                                        "receive_threshold": {
                                                "reporting_enabled": true,
                                                "level_db": -35
                                        },
                                }
                        }
                ]
},
{
                "name": "module_2",
                "type": "476-153",
                "configuration": {
                "enabled": true
                },
                "ports": [
                        {
                                "channelID": "-334049038",
                                "channelName": "port_3",
                                "channelType": "2",
                                "configuration": {
                                        "channel_select_position": 1
                                        "api_provides_ringback": false,

                                        "lp_filter_300hz_enabled": false,
                                        "hp_filter_300hz_enabled": false,
                                        "custom_filter_enabled": false,
                                        "snr_enabled": false,
                                        "ptt_polarity_normal": true,
                                        "cor_polarity_normal": true,
                                        "input_impedance_600": true,
                                        "transmit_level_gain_db": 0,
                                        "receive_level_gain_db": 0,
                                        "audio_delay": {
                                                "receive_ticks": 0,
                                                "transmit_ticks": 1000
                                        },
                                        "transmit_threshold": {
                                                "reporting_enabled": true,
```

```
                                        "level_db": -35
                                        },
                                "receive_threshold": {
                                        "reporting_enabled": true,
                                        "level_db": -35
                                },
                        },
                },
                {
                        "channelID": "517294308",
                        "channelName": "port_4",
                        "channelType": "2",
                        "configuration": {
                                "channel_select_position": 1
                                "api_provides_ringback": false,
                                "lp_filter_300hz_enabled": false,
                                "hp_filter_300hz_enabled": false,
                                "custom_filter_enabled": false,
                                "snr_enabled": false,
                                "ptt_polarity_normal": true,
                                "cor_polarity_normal": true,
                                "input_impedance_600": true,
                                "transmit_level_gain_db": 0,
                                "receive_level_gain_db": 0,
                                "audio_delay": {
                                        "receive_ticks": 0,
                                        "transmit_ticks": 1000
                                },
                                "transmit_threshold": {
                                        "reporting_enabled": true,
                                        "level_db": -35
                                },
                                "receive_threshold": {
                                        "reporting_enabled": true,
                                        "level_db": -35
                                },
                        },
                }
        ]
},
{
        "name": "module_3",
        "type": "476-153",
        "configuration": {
                "enabled": true
        },
        "ports": [
                {
                        "channelID": "766242398",
                        "channelName": "port_5",
                        "channelType": "2",
                        "configuration": {
                                "channel_select_position": 1
                                "api_provides_ringback": false,
                                "lp_filter_300hz_enabled": false,
                                "hp_filter_300hz_enabled": false,
                                "custom_filter_enabled": false,
                                "snr_enabled": false,
                                "ptt_polarity_normal": true,
                                "cor_polarity_normal": true,
                                "input_impedance_600": true,
                                "transmit_level_gain_db": 0,
                                "receive_level_gain_db": 0,
                                "audio_delay": {
                                        "receive_ticks": 0,
                                        "transmit_ticks": 1000
                                },
                                "transmit_threshold": {
                                        "reporting_enabled": true,
                                        "level_db": -35
                                },
                                "receive_threshold": {
                                        "reporting_enabled": true,
                                        "level_db": -35
                                },
```

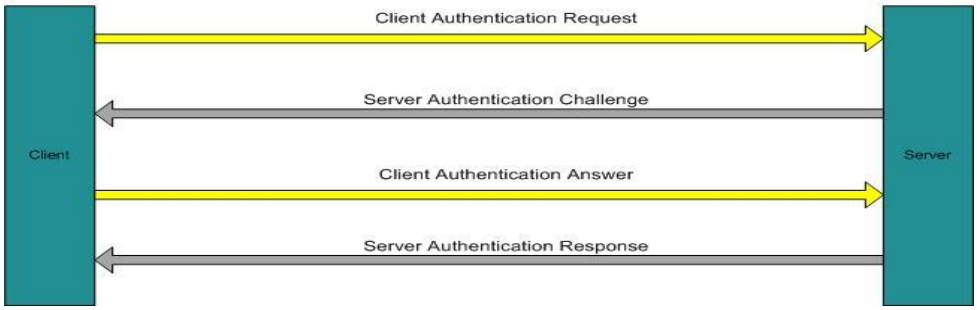
```
                                        },
                                },
                                {
                                        "channelID": "-537012222",
                                        "channelName": "port_6",
                                        "channelType": "2",
                                        "configuration": {
                                                "channel_select_position": 1
                                                “api_provides_ringback”: false,
                                                "lp_filter_300hz_enabled": false,
                                                "hp_filter_300hz_enabled": false,
                                                "custom_filter_enabled": false,
                                                "snr_enabled": false,
                                                "ptt_polarity_normal": true,
                                                "cor_polarity_normal": true,
                                                "input_impedance_600": true,
                                                "transmit_level_gain_db": 0,
                                                "receive_level_gain_db": 0,
                                                "audio_delay": {
                                                        "receive_ticks": 0,
                                                        "transmit_ticks": 1000
                                                },
                                                "transmit_threshold": {
                                                        "reporting_enabled": true,
                                                        "level_db": -35
                                                },
                                                "receive_threshold": {
                                                        "reporting_enabled": true,
                                                        "level_db": -35
                                                },
                                        },
                                }
                        ]
                },
                {
                        "name": "module_4",
                        "type": "476-777",
                        "configuration": {
                        },
                        "ports": [
                                {
                                        "name": "port_1",
                                        "configuration": {
                                                "channel_select_position": 1
                                                “api_provides_ringback”: false,
                                                "ptt_polarity_normal": true,
                                                "cor_polarity_normal": true,
                                                "input_impedance_600": true,
                                                "transmit_level_gain_db": 0,
                                                "receive_level_gain_db": 0,
                                        }
                                }
                        ]
                }
        ]
}
```

XI. API Authentication to the SVoIP Gateway

To successfully connect to a gateway you must authenticate after you connect. The process is as follows:

1. The client (or third party app) issues a request
2. The server will issue a random 64 bit challenge code
3. The client encodes the 64 bit challenge code and returns it as an answer to the challenge
4. The server verifies the answer and sends a response of either success or failure

Client Authentication



TEA is used for the encryption with the 128 bit key **0x606f686e44656e6e427265744a617904** and the delta that is used in the TEA algorithm is **0x9e3779b9.**

Example De/Encryption Algorithms

```
/* encrypt
 *   Encrypt 64 bits with a 128 bit key using TEA
 *   From http://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm
 * Arguments:
 *   v - array of two 32 bit uints to be encoded in place
 *   k - array of four 32 bit uints to act as key
 * Returns:  *   v - encrypted result

void TeaEncryption (u32* v, u32* k) {
   u32 v0=v[0], v1=v[1], sum=0, i;                /* set up */
   u32 delta=0x9e3779b9;                          /* a key schedule constant */
   u32 k0=k[0], k1=k[1], k2=k[2], k3=k[3];        /* cache key */
   for (i=0; i < 32; i++) {                       /* basic cycle start */
      sum += delta;
      v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
      v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
   } /* end cycle */
   v[0]=v0; v[1]=v1;
}
void TeaDecryption (u32* v, u32* k) {
   u32 v0=v[0], v1=v[1], sum=0xC6EF3720, i;    /* set up */
   u32 delta=0x9e3779b9;                          /* a key schedule constant */
   u32 k0=k[0], k1=k[1], k2=k[2], k3=k[3];        /* cache key */
   for (i=0; i<32; i++) {                         /* basic cycle start */
      v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
      v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
      sum -= delta;
   }                                    /* end cycle */
   v[0]=v0; v[1]=v1;}
```

XII. API Checksum Calculation

The checksums that we use are based on the CRC-16-CCITT standard. It uses a polynomial of 0x1021 for calculation.

Example Checksum Algorithm

```
#define POLY 0x8408
/*
//                          16   12   5
// this is the CCITT CRC 16 polynomial X  + X  + X  + 1.
// This works out to be 0x1021, but the way the algorithm works
// lets us use 0x8408 (the reverse of the bit pattern).  The high
// bit is always assumed to be set, thus we only use 16 bits to
// represent the 17 bit value.
*/

u16 crc16(char *data_p, u16 length)
{
    char i;
    int data;
    int crc = 0xffff;

    if (length == 0)
        return (~crc);
    do
    {
        for (i=0, data=(int)0xff & *data_p++;   i < 8;  i++, data >>= 1)
        {
            if ((crc & 0x0001) ^ (data & 0x0001)) crc = (crc >> 1) ^ POLY;
            else  crc >>= 1;
        }
    } while (--length);

    crc = ~crc;
    data = crc;
    crc = (crc << 8) | (data >> 8 & 0xff);

    return (crc);
}
```