Clearone Converge Pro 2

Extron Python Module Help

Module Overview

DSP systems have become very complicated and highly configurable. The control system programmer tends to be asked to do almost anything when it comes to system control. The problem comes in when the control system modules are bloated with functionality and extra controls points that the system must deal with, even when those control points aren't needed.

It has become a standard practice in DSP control to break module design up into components handling very specific functionality, such as volume control. If you only need to manage 10 volume controls, the control system programmer can include 10 instances of the volume control component and nothing else. Because the components have very specific reduced functionality and only those items that need to be controlled will be included, this keeps the overhead and footprint at the minimum required from system to system.

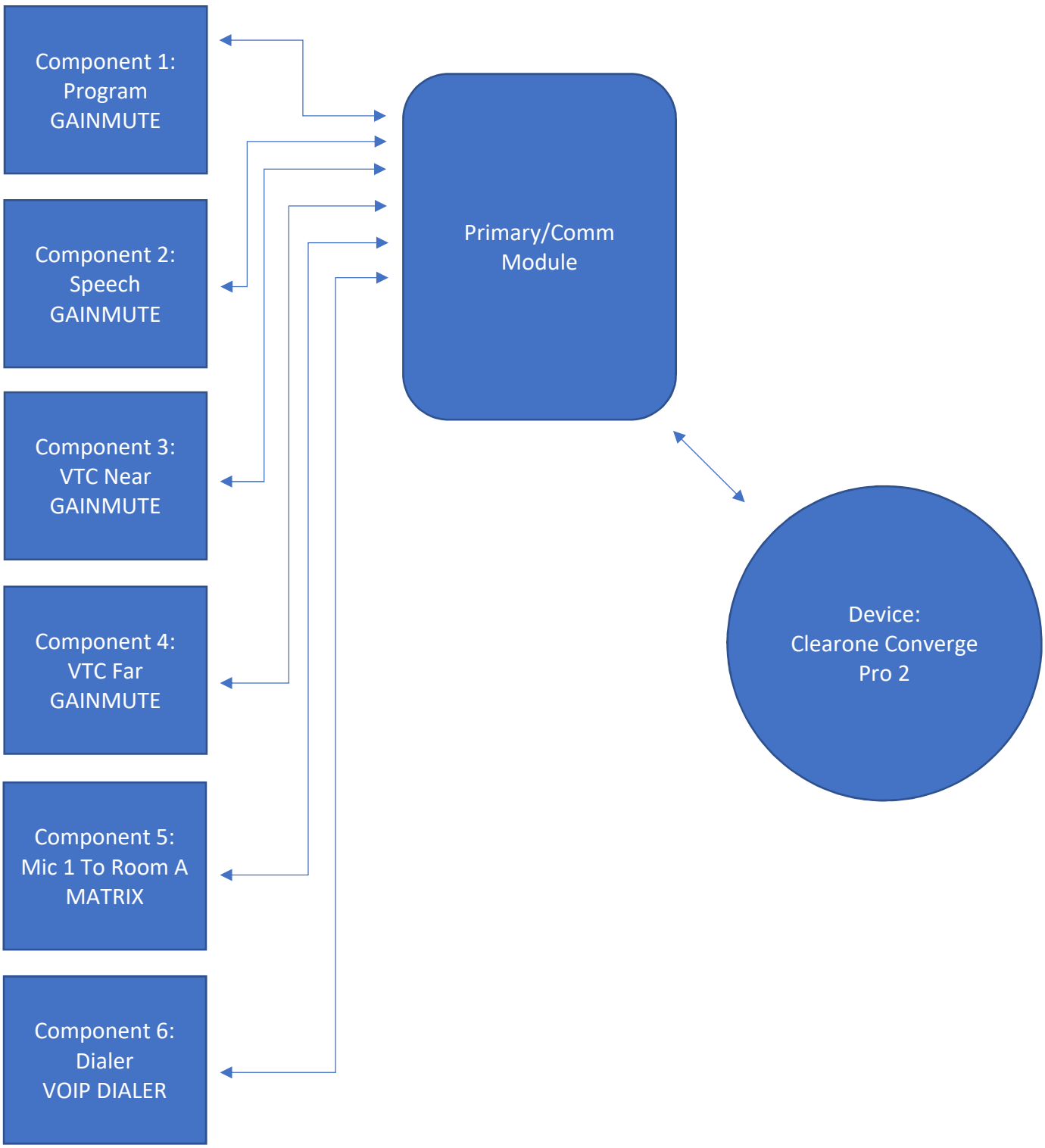
The design of this Extron Python module follows this component model and enables all communication to work using a single TCP connection or RS232 port. Each instantiated primary/comm module is responsible for controlling a single ClearOne Converge Pro 2 unit. The desired components are mapped to the primary/comm module by the Extron programmer when initializing the module. Multiple primary/comm modules can be included in your program to control multiple ClearOne units, if desired. Each primary/comm module can have a virtually unlimited number of components mapped to it. The only limitation will be the memory available in the Extron processor itself.

A demo program (*Clearone_Converge_Pro_2_Demo.gs*) showing how to use the module has been included. A separate demo program has been created for Skype For Business functionality (*Clearone_Converge_Pro_2_Skype_Demo.gs*). In addition, a .pccp2 file (*ExtronDemo.pccp2*) was created for Extron testing purposes and MUST be used for proper operation of the demo program.

Once the primary/comm module has determined that it is communicating with the Converge Pro 2 unit, it will initialize all individual components that have been registered to it. Once a component receives all the responses it is looking for, it will instruct the processing module that its initialization is complete. The primary/comm module will then request the next component to initialize. Once all components are initialized, the primary/comm module will be fully initialized and ready for full control of all mapped components.

Important Note: in order for the module to work correctly, ALL components must be configured correctly and have corresponding DSP control points in the DSP program.

Example of how the components work within the Extron module to talk to the Clearone Converge Pro 2:



Implementation

To use the ClearOne Converge Pro 2 module, the Extron programmer must perform the following steps:

1) Add the *Clearone_Converge_Pro_2_v1_0_0.py* file to the GlobalScripter workspace.

2) Import the *Clearone_Converge_Pro_2_v1_0_0.py* into the main program.

   Example
   import Clearone_Converge_Pro_2_v1_0_0 as MyClearOne

3) Instantiate the module using either Ethernet or Serial communication and set any required credentials.

   Serial Example
   MyModule = MyClearOne.SerialClass(Host=IPCPPro350, Port='COM1', Baud=57600, Data=8, Parity='None', Stop=1, FlowControl='Off', CharDelay=0, Mode='RS232')

   Ethernet Example
   MyModule = MyClearOne.EthernetClass(Hostname='192.168.187.54', IPPort=23)
   MyModule.SetCredentials(Username='clearone', Password='converge')

4) Map any required components to the main module using "Add" methods.

   Examples
   MyVolume1 = MyModule.AddGainMuteComponent('GAIN_FINE', 'Mic-01', 5)
   MyIn1Out1 = MyModule.AddMatrixComponent('Mic-01', 'Record', 'Gated', 2)
   MyADialer = MyModule.AddAnalogDialerComponent('TELCO_RX 201')

5) Call the "Reinit" method on the main module to establish communication with the device and begin the initialization process.

   Example
   MyModule.Reinit()

6) Once initialization is complete, control the module components directly by calling specified methods. Feedback is provided in two ways: a) via direct query methods that return the data requested and b) via unsolicited events that can be subscribed to in the main program.

   Control Example
   MyVolume1.SetMute(1)
   MyIn1Out1.ToggleCrosspoint()
   MyADialer.DialNumber('8675309')

   Direct Query FB Example
   myMuteState = MyVolume1.GetMute()
   myHookStatus = MyADialer.GetHookStatus()

   Asynchronous Event FB Example
   @event(MyVolume1, 'OnMuteStateChange')
   @event(MyADialer, 'OnAnalogIncomingRingChange')

Further details regarding many of the above items can be found later in this document.

Important Note on Data Access

As there is no way (currently) to encapsulate/obfuscate modules within Global Scripter, the code is readily available for viewing and editing. This presents a problem in regards to hiding variables and methods that the module needs to function but you as a programmer should not edit directly. The module contains a great deal of data that should not be edited or accessed directly, as doing so may cause the module to stop functioning.

As such, please access the module from your program using ONLY the methods described below and DO NOT attempt to directly change any internal member variables or directly call methods that are not listed below.

Failure to heed this warning may result in undesired operation and behavior.

Primary/Comm Module

The primary/comm module must be added to your program prior to adding any components. To add to your program, instantiate a new object with a type of either SerialClass or EthernetClass object from the module, passing in the parameters as shown in the example below.

Serial Example
MyModule = MyClearOne.SerialClass(Host=IPCPPro350, Port='COM1', Baud=57600, Data=8, Parity='None', Stop=1, FlowControl='Off', CharDelay=0, Mode='RS232')

Ethernet Example
MyModule = MyClearOne.EthernetClass(Hostname='192.168.187.54', IPPort=23)

Available Control/Feedback

The following methods and events are available for your use:

Commands

- SetCredentials([Username], [Password])
  - Set the username and password, if required
  - Username = login username (optional, set if needed using keyword...if not needed, leave blank...will default to '')
  - Password = login password (optional, set if needed using keyword...if not needed, leave blank...will default to '')

- SetDebug(State)
  - Set whether to print internal module debug statements to Trace window. Useful if debugging. Should be set to Off (default) if not actively debugging as it generates a lot of messages.
  - State = integer (0 = Off | 1 = On)

- SetPoll(State)
  - Set whether to have all components poll for current values every 15 seconds. As most components will provide this information automatically when they change, this should normally be set to Off (default) unless needed for a specific reason as it generates a lot of messages to the device.
  - State = integer (0 = Off | 1 = On)

- ConnectToDevice()
  - Manually connect to the device. This will be triggered automatically when Reinit() is called and when communication is lost so will usually not be needed but has been included in case there is need to manually connect.

- Reinit()
  - Connect to device and reinitialize the state of all components. Should be triggered within program Initialize() method.

Direct Feedback (Request)

- IsInitializedCheck()
  - Returns value indicating if the module is initialized. Once it is, the module is ready to use.
  - Return = integer (0 = Not Initialized | 1 = Initialized)

Event Feedback (Unsolicited)

- Connected(Interface, State)
  - Indicates an IP connection with the device has been established.
  - Interface = the object that triggered the event
  - State = 'Connected'

- Disconnected(Interface, State)
  - Indicates an IP connection with the device has been lost.
  - Interface = the object that triggered the event
  - State = 'Disconnected'

- ModuleInfo(Interface, Communicating, Initialized)
  - Provides information on whether module has received data from the device (Communicating) and whether the module has received all relevant data from the device that it needs to operate correctly (Initialized)
  - Interface = the object that triggered the event
  - Communicating = integer (0 = Not Communicating | 1 = Communicating)
  - Initialized = integer (0 = Not Initialized | 1 = Initialized)

Components

The following components are available for use. The control and feedback capabilities of each will be detailed further below.

1) EPGeneric
2) RoomGeneric
3) GainMute
4) Matrix
5) Crosspoint
6) Preset
7) Macro
8) AnalogDialer
9) VoIPDialer
10) SkypeDialer

EPGeneric Component

The EPGeneric component was created in order to allow for controlling “End Point (EP)” objects within the Converge Pro 2 program that may not be controllable using one of the other standard components.

The component is capable of controlling an end point Clearone object via either Digital, Analog, or Serial methods. Multiple EPGeneric components can be added to your programming to allow for a fully flexible solution to match any programming problem you might encounter.

Add Component

- AddEPGenericComponent(SignalType, EndPointName, BlockNumber, ParameterName, [IsPollable])
    - SignalType = 'Digital', 'Analog', or 'Serial' (string, case sensitive)
    - EndPointName = name of endpoint in program (string, case-sensitive)
        - *Examples: ‘Mic-01’, ‘VoIP_1_Rx’*
    - BlockNumber = Any valid End Point block name per protocol (string, case-sensitive)
        - *Examples: ‘LEVEL, ‘AEC’, ‘GATING’*
    - ParameterNumber = Any valid parameter name for the given block per protocol (string, case-sensitive)
        - *Examples: ‘PHAN_PWR, ‘GAIN_COURSE, ‘ENABLE’*
    - IsPollable = flag for whether or not object provides information on query (Optional. Set if needed using keyword. If not needed, leave blank. Will default to True)

Example
MyDialerPrivacy = MyModule.AddEPGenericComponent('Digital', 'Dialer_1_Tx', 'LEVEL', 'MUTE', IsPollable=False)

Available Control/Feedback

The following methods and events are available for your use:

Commands

- SetDigital(State)
    - Set the state of the control point to on or off
    - State = integer (0 = Off | 1 = On)
    - Only applicable if component SignalType is ‘Digital’

- ToggleDigital()
    - Toggle the state of the control point
    - Only applicable if component SignalType is ‘Digital’

- SetAnalog(Value)
    - Set the value of the control point
    - Value = integer or float
    - Only applicable if component SignalType is ‘Analog’

- SetSerial(Value)
    - Set the value of the control point
    - Value = string
    - Only applicable if component SignalType is ‘Serial’

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

- GetDigital()
  - Returns the current state of the control point
  - Return = integer (0 = Off | 1 = On)
  - Only applicable if component SignalType is 'Digital'

- GetAnalog()
  - Returns the current value of the control point
  - Return = float (convert to integer in program if needed)
  - Only applicable if component SignalType is 'Analog'

- GetSerial()
  - Returns the current value of the control point
  - Return = string
  - Only applicable if component SignalType is 'Serial

Event Feedback (Unsolicited)

- OnEPDigitalStateChange(Interface, Value)
  - Provides the current state of the control point
  - Interface = the object that triggered the event
  - Value = integer (0 = Off | 1 = On)
  - Will only fire if component SignalType is 'Digital'

- OnEPAnalogStateChange(Interface, Value)
  - Returns the current value of the control point
  - Interface = the object that triggered the event
  - Value = float (convert to integer in program if needed)
  - Will only fire if component SignalType is 'Analog'

- OnEPSerialStateChange(Interface, Value)
  - Returns the current value of the control point
  - Interface = the object that triggered the event
  - Value = string
  - Will only fire if component SignalType is 'Serial

RoomGeneric Component

The RoomGeneric component was created in order to allow for controlling "Room" objects within the Converge Pro 2 program that may not be controllable using one of the other standard components.

The component is capable of controlling an end point Clearone object via either Digital, Analog, or Serial methods. Multiple RoomGeneric components can be added to your programming to allow for a fully flexible solution to match any programming problem you might encounter.

Add Component

- AddRoomGenericComponent(SignalType, RoomNumber, Option, [P1], [P2])
    - SignalType = 'Digital', 'Analog', or 'Serial' (string, case sensitive)
    - RoomNumber = room number in program (integer, 1 -255)
    - Option = Any valid option name per protocol (string, case-sensitive)
        - *The keys in 'RoomOptions' dictionary shown at end of document contains valid values.*
    - P1/P2 = Optional. Use if the Option you've chosen requires either value (per protocol). Set if needed using keyword. If not needed, leave blank.)

Example
MyWall1 = MyModule.AddRoomGenericComponent('Digital', 1, 'IndividualDividerState', P1=1)

Available Control/Feedback

The following methods and events are available for your use:

Commands

- SetDigital(State)
    - Set the state of the control point to on or off
    - State = integer (0 = Off | 1 = On)
    - Only applicable if component SignalType is 'Digital'

- ToggleDigital()
    - Toggle the state of the control point
    - Only applicable if component SignalType is 'Digital'

- SetAnalog(Value)
    - Set the value of the control point
    - Value = integer or float
    - Only applicable if component SignalType is 'Analog'

- SetSerial(Value)
    - Set the value of the control point
    - Value = string
    - Only applicable if component SignalType is 'Serial'

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

- GetDigital()
  - Returns the current state of the control point
  - Return = integer (0 = Off | 1 = On)
  - Only applicable if component SignalType is 'Digital'

- GetAnalog()
  - Returns the current value of the control point
  - Return = float (convert to integer in program if needed)
  - Only applicable if component SignalType is 'Analog'

- GetSerial()
  - Returns the current value of the control point
  - Return = string
  - Only applicable if component SignalType is 'Serial

Event Feedback (Unsolicited)

- OnRoomDigitalStateChange(Interface, Value)
  - Provides the current state of the control point
  - Interface = the object that triggered the event
  - Value = integer (0 = Off | 1 = On)
  - Will only fire if component SignalType is 'Digital'

- OnRoomAnalogStateChange(Interface, Value)
  - Returns the current value of the control point
  - Interface = the object that triggered the event
  - Value = float (convert to integer in program if needed)
  - Will only fire if component SignalType is 'Analog'

- OnRoomSerialStateChange(Interface, Value)
  - Returns the current value of the control point
  - Interface = the object that triggered the event
  - Value = string
  - Will only fire if component SignalType is 'Serial

GainMute Component

This component controls all Gain/Mute points in the Clearone Converge Pro 2 where the BlockName is 'LEVEL' and the ParameterName is 'GAIN' or 'GAIN_FINE'. The control point must also have a ParameterName of 'MUTE'. This module also uses MAX_GAIN and MIN_GAIN to automatically set the limits.

If the control point that you need to control does not meet with these requirements, use the EPGeneric component instead.

Add Component

- **AddGainMuteComponent(GainType, EndPointName, StepValue)**
  - GainType = 'GAIN' or 'GAIN_FINE' (string, case sensitive)
    - *You must pick 'GAIN_FINE' for microphones. All others should be 'GAIN'.*
  - EndPointName = name of endpoint in program (string, case-sensitive)
    - *Examples: 'Mic-01', 'Speakers'*
  - StepValue = amount to increment/decrement when ramping in dB (integer)

Examples
MyVolume1 = MyModule.AddGainMuteComponent('GAIN_FINE', 'Mic-01', 5)
MyVolume2 = MyModule.AddGainMuteComponent('GAIN', 'Speakers', 5)

Available Control/Feedback

The following methods and events are available for your use:

Commands

- **SetLevelPercent(Value)**
  - Set the percent value of the gain level
  - Value = integer (0 – 100)

- **SetLevelDB(Value)**
  - Set the dB value of the gain level
  - Value = float (between MIN_GAIN and MAX_GAIN for EndPoint being controlled)

- **Raise()**
  - Increment the gain level by value of StepSize
  - Will continue until Stop() is called or MAX_GAIN has been reached.

- **Lower()**
  - Decrement the gain level by value of StepSize
  - Will continue until Stop() is called or MIN_GAIN has been reached.

- **Stop()**
  - Stop ramping the gain level.

- **SetMute(State)**
  - Set the mute state to on or off
  - State = integer (0 = Off | 1 = On)

- ToggleMute()
    - Toggle the mute state

Direct Feedback (Request)

- IsInitialized()
    - Returns value indicating if the component is initialized (useful for debugging)
    - Return = integer (0 = Not Initialized | 1 = Initialized)

- GetLevelPercent()
    - Returns the current percent value of the gain level
    - Return = integer (0 – 100)

- GetLevelDB()
    - Returns the current dB value of the gain level
    - Return = float (between MIN_GAIN and MAX_GAIN for EndPoint being controlled)

- GetMute()
    - Returns the current mute state
    - Return = integer (0 = Off | 1 = On)

Event Feedback (Unsolicited)

- OnLevelDBChange(Interface, Value)
    - Provides the current dB value of the gain level
    - Interface = the object that triggered the event
    - Value = float (between MIN_GAIN and MAX_GAIN for EndPoint being controlled)

- OnLevelPercentChange(Interface, Value)
    - Provides the current percent value of the gain level
    - Interface = the object that triggered the event
    - Value = integer (0 - 100)

- OnMuteStateChange(Interface, State)
    - Returns the current mute state
    - Interface = the object that triggered the event
    - Value = integer (0 = Off | 1 = On)

Matrix Component

This module controls the MT protocol command and is used to control a single cross point on the DSP Matrix. Provide as many of these as you need for each cross point needed to control.

Add Component

- AddMatrixComponent(InputEndPointName, OutputEndPointName, CrosspointType, StepValue)
  - InputEndPointName = name of input endpoint in program (string, case-sensitive)
    - *Examples: 'Mic-01', 'VoIP_1_Rx'*
  - OutputEndPointName = name of output endpoint in program (string, case-sensitive)
    - *Examples: 'Speakers', 'VoIP_1_Tx'*
  - CrosspointType = Any valid crosspoint type name per protocol (string, case-sensitive)
    - *The keys in 'CrosspointTypes' dictionary shown at end of document contains valid values.*
  - StepValue = amount to increment/decrement when ramping in dB (integer)

Examples
MyIn1Out1 = MyModule.AddMatrixComponent('Mic-01', 'Record', 'Gated', 2)
MyIn3Out1 = MyModule.AddMatrixComponent('Fader_1', 'Record', 'Crosspoint', 2)

Available Control/Feedback

The following methods and events are available for your use:

Commands

- SetCrosspoint(State)
  - Set the crosspoint mute to on or off
  - State = integer (0 = Off | 1 = On)
  - When State = 1: If not routed, will route based on crosspoint type. If already routed, will unmute the crosspoint, if muted.
  - When State = 0: If not routed, will route based on crosspoint type and mute the crosspoint.

- ToggleCrosspoint()
  - Toggle the crosspoint mute state

- ClearCrosspoint()
  - Unroute and clear the crosspoint with 0 dB attenuation.

- SetAttenuationPercent(Value)
  - Set the percent value of the attenuation level
  - Value = integer (0 – 100)

- SetAttenuationDB(Value)
  - Set the dB value of the attenuation level
  - Value = float (between MIN_GAIN and MAX_GAIN for EndPoint being controlled)

- Raise()
  - Increment the attenuation level by value of StepSize
  - Will continue until Stop() is called or MAX_GAIN has been reached.

- Lower()
  - Decrement the attenuation level by value of StepSize
  - Will continue until Stop() is called or MIN_GAIN has been reached.

- Stop()
  - Stop ramping the attenuation level.

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

- GetAttenuationPercent()
  - Returns the current percent value of the attenuation level
  - Return = integer (0 – 100)

- GetAttenuationDB()
  - Returns the current dB value of the attenuation level
  - Return = float (between MIN_GAIN and MAX_GAIN for EndPoint being controlled)

- GetCrosspoint()
  - Returns the current crosspoint state
  - Return = integer (0 = Off | 1 = On)

Event Feedback (Unsolicited)

- OnAttenuationDBChange(Interface, Value)
  - Provides the current dB value of the attenuation level
  - Interface = the object that triggered the event
  - Value = float (between MIN_GAIN and MAX_GAIN for EndPoint being controlled)

- OnAttenuationPercentChange(Interface, Value)
  - Provides the current percent value of the attenuation level
  - Interface = the object that triggered the event
  - Value = integer (0 - 100)

- OnCrosspointChange(Interface, State)
  - Returns the current crosspoint state
  - Interface = the object that triggered the event
  - Value = integer (0 = Off | 1 = On)

Preset Component

This component controls room Preset recall for the Clearone Converge Pro 2. Each instance of this component supports a single room and partition. Provide as many other instances for each room and partition that you need.

Add Component

- AddPresetComponent(RoomNumber, RoomPartitionName)
    - RoomNumber = room number in program (integer, 1 -255)
    - RoomPartitionName = parameter that is used to set the name of the room partition to use (string, case-sensitive)
        - *Example: 'Part_A'*

Example
MyPresets = MyModule.AddPresetComponent(1, 'Part_A')

Available Control/Feedback

The following methods and events are available for your use:

Commands

- RecallPreset(Name)
    - Set the preset to recall
    - Name = string

Direct Feedback (Request)

- IsInitialized()
    - Returns value indicating if the component is initialized (useful for debugging)
    - Return = integer (0 = Not Initialized | 1 = Initialized)

Macro Component

This component controls macro recall for the Clearone Converge Pro 2. A single instance of this component will allow you to recall an unlimited number of macros from the DSP program.

Add Component

- AddMacroComponent()

Example
MyMacros = MyModule.AddMacroComponent()

Available Control/Feedback

The following methods and events are available for your use:

Commands

- RunMacro(Name)
  - Set the macro to run
  - Name = string

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

AnalogDialer Component

This module control the TELCO_RX block names and parameters associated with the analog dialing features.

Add Component

- AddAnalogDialerComponent(EndPointName)
    - EndPointName = name of endpoint in program (string, case-sensitive)
        - *Example: 'Dialer_1'*

Examples
MyDialer = MyModule.AddAnalogDialerComponent('TELCO_RX 201')

Available Control/Feedback

The following methods and events are available for your use:

Commands

- OnHook()
    - Set the hook state to On
- OffHook()
    - Set the hook state to Off
- HookFlash()
    - Flash the hook state
- Redial()
    - Redial the last number called
- DigitPressed(Digit)
    - Send a single digit press to the device
    - Value = string
    - Only digits (0-9) and */# allowed
    - Module will also build a keypad string from digits passed in which is used with the Dial() command. If there is an active call, it will also send DTMF tones as appropriate.
- DigitReleased(Digit)
    - Send a single digit release to the device
    - Value = string
    - Only digits (0-9) and */# allowed
- ClearKeypadText()
    - Clear the internally stored keypad text
- BackSpaceKeypadText()
    - Remove a character from the internally stored keypad text

- DialNumber(Number)
  - Dial a number
  - Value = string
  - Only digits (0-9) and */# allowed

- Dial()
  - Dial the currently stored keypad string

- SetAutoAnswerRingCount(Count)
  - Set the number of rings for Auto Answer
  - Count = integer (0 – 3)
  - Setting the Count to 0 will turn off Auto Answer

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

- GetHookStatus()
  - Returns the current hook status
  - Return = integer (0 = On Hook | 1 = Off Hook)

- GetAutoAnswerRingCount()
  - Returns the current Auto Answer ring count
  - Return = integer (0 – 3)

- GetCallDuration()
  - Returns the duration of the last call
  - Return = integer

- GetCallerIDInformation()
  - Returns the current Caller ID information
  - Return = a 'Caller ID' object which consists of:
    - Time (string)
    - Date (string)
    - PhoneNumber (string)
    - Name (string)

- GetLastNumberDialed()
  - Returns the last number dialed
  - Return = string

- GetLocalNumber()
  - Returns the local number
  - Return = string

Event Feedback (Unsolicited)

- OnAnalogKeypadTextChange(Interface, Value)
  - Provides the current keypad string stored in the module
  - Interface = the object that triggered the event
  - Value = string

- OnAnalogAutoAnswerChange(Interface, Value)
  - Provides the current Auto Answer ring count
  - Interface = the object that triggered the event
  - Return = integer (0 – 3)

- OnAnalogCallDurationChange(Interface, Value)
  - Provides the duration of the last call
  - Interface = the object that triggered the event
  - Return = integer

- OnAnalogCallerIDNotification(Interface, Value)
  - Provides the current Caller ID information
  - Interface = the object that triggered the event
  - Return = a 'Caller ID' object which consists of:
    - Time (string)
    - Date (string)
    - PhoneNumber (string)
    - Name(string)

- OnAnalogHookChange(Interface, Value)
  - Provides the current hook status
  - Interface = the object that triggered the event
  - Return = integer (0 = On Hook | 1 = Off Hook)

- OnAnalogIncomingRingChange(Interface, Value)
  - Provides information on whether there is an incoming call
  - Interface = the object that triggered the event
  - Return = integer (0 = Not Ringing | 1 = Ringing)

- OnAnalogLastNumberDialedChange(Interface, Value)
  - Provides information on the last number dialed
  - Interface = the object that triggered the event
  - Return = string

- OnAnalogLocalNumberChange(Interface, Value)
  - Provides information on the local number
  - Interface = the object that triggered the event
  - Return = string

VoIP Dialer Component

This module control the UA block names and parameters associated with the VoIP dialing features.

Add Component

- AddVoIPDialerComponent(EndPointName)
    - EndPointName = name of endpoint in program (string, case-sensitive)
        - *Example: 'Voip_1'*

Examples
MyDialer = MyModule.AddVoIPDialerComponent('Voip_1')

Available Control/Feedback

The following methods and events are available for your use:

Commands

- SelectLine(Value)
    - Select a line
    - Value = integer (1 – 5)
    - If the line is idle, you will get a dial tone. If the line is on hold, it will re-acquire. If the line is active, it will hang up the call.

- DigitPressed(Digit)
    - Send a single digit press to the device
    - Value = string
    - Only digits (0-9) and */# allowed
    - Module will also build a keypad string from digits passed in which is used with the Dial() command. If there are active lines, it will also send DTMF tones as appropriate.

- DigitReleased(Digit)
    - Send a single digit release to the device
    - Value = string
    - Only digits (0-9) and */# allowed

- ClearKeypadText()
    - Clear the internally stored keypad text

- BackSpaceKeypadText()
    - Remove a character from the internally stored keypad text

- DialNumber(Number)
    - Dial a number directly
    - Value = string
    - Only digits (0-9) and */# allowed

- Dial()
    - Dial the currently stored keypad string

- Reject(Line)
    - Reject an incoming call on a specific line
    - Line = integer (1 – 5)

- Hold()
    - Put selected line on hold

- Transfer()
    - Start the transfer process on the selected line
    - For reference, a users next step after this is to use the keypad to enter the extension to transfer to. Once there is a connection with the extension, they would have to send/press Transfer again.

- BlindTransfer()
    - Start the blind transfer process on the selected line
    - For reference, a users next step after this is to use the keypad to enter the extension to transfer to. Once there is a connection with the extension, the call will transfer immediately.

- Conference(Line)
    - Add a line which has been place on hold to the conference
    - Line = integer (1 – 5)
    - For reference, this will put your current call on hold and you will have to send Conference on all lines to finish setting up the conference.

- Redial()
    - Redial the last number called

- MuteOn()
    - Mute the active line
    - This is NOT the same as Transmit Level Mute

- MuteOff()
    - Unmute the active line
    - This is NOT the same as Transmit Level Mute

- OnHook()
    - Hang up the selected call(s)

- OffHook()
    - Pick up a call or start a call

- HookToggle()
    - Hang up/pick up depending on current hook status

- CallForwardDisable()
    - End call forward mode

- CallForwardEnable()
  - Start the process of enabling 'UNCONDITIONAL' call forward.
  - For reference, a users next step after this is to use the keypad to enter the extension to forward calls to. Once this happened correctly, you will be in call forward mode.

- CallForwardBusy()
  - Start the process of enabling 'BUSY' call forward.
  - For reference, a users next step after this is to use the keypad to enter the extension to forward calls to. Once this happened correctly, you will be in call forward mode.

- CallForwardNoAnswer()
  - Start the process of enabling 'NO_REPLY' call forward.
  - For reference, a users next step after this is to use the keypad to enter the extension to forward calls to. Once this happened correctly, you will be in call forward mode.

- DNDDisable()
  - Turn off DND mode

- DNDCallMute()
  - Turn on DND 'CALL_MUTE' mode

- DNDCallReject()
  - Turn on DND 'CALL_REJECT' mode

- ClearErrors()
  - Reset the current errors, if any

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

- GetActiveLine()
  - Returns the currently active line
  - Return = integer (0 - 5)

- GetErrors()
  - Returns the current Errors information
  - Return = an 'Errors' object which consists of:
    - ErrorText1 (string)
    - ErrorText2 (string)
    - ErrorText3 (string)
    - ErrorText4 (string)

- GetStatus()
    - Returns the current Status information
    - Return = a 'Status' object which consists of:
        - HoldStatus (integer)
        - MuteStatus (integer)
        - RingBackStatus (integer)
        - RingingStatus (integer)
        - CallForwardState (integer)
        - CallForwardStatus (integer)
        - CallForwardStatusText (string)
        - DNDStatus (integer)
        - DNDStatusText (string)
        - BusyStatus (integer)
        - WarningErrorStatus (integer)
        - CallWaiting (integer)
        - LineStatus[] – list of 5 LineStatus object, each of which consist of:
            - LineLEDState (integer)
            - LineStatus (integer)
            - LineStatusText (string)
            - LineInfo (string)
            - LineID (integer)

Event Feedback (Unsolicited)

- OnVoIPKeypadTextChange(Interface, Value)
    - Provides the current keypad string stored in the module
    - Interface = the object that triggered the event
    - Value = string

- OnVoIPErrorChange(Interface, Value)
    - Provides the current Status information
    - Interface = the object that triggered the event
    - Return = an 'Errors' object which consists of:
        - ErrorText1 (string)
        - ErrorText2 (string)
        - ErrorText3 (string)
        - ErrorText4 (string)

- OnVoIPStatusChange(Interface, Value)
    - Provides the current Status information
    - Interface = the object that triggered the event
    - Return = a 'Status' object which consists of:
        - HoldStatus (integer)
        - MuteStatus (integer)
        - RingBackStatus (integer)
        - RingingStatus (integer)
        - CallForwardState (integer)
        - CallForwardStatus (integer)
        - CallForwardStatusText (string)
        - DNDStatus (integer)
        - DNDStatusText (string)
        - BusyStatus (integer)
        - WarningErrorStatus (integer)
        - CallWaiting (integer)
        - LineStatus[] – list of 5 LineStatus object, each of which consist of:
            - LineLEDState (integer)
            - LineStatus (integer)
            - LineStatusText (string)
            - LineInfo (string)
            - LineID (integer)

Skype Component

This module control the UA block names and parameters associated with the Skype for Business dialing features.

Add Component

- AddSkypeComponent(EndPointName)
    - EndPointName = name of endpoint in program (string, case-sensitive)
        - *Example: 'Skype_Name_2_01'*

Examples
MySkype = MyModule.AddSkypeComponent('Skype_Name_2_01')

Available Control/Feedback

The following methods and events are available for your use:

Commands

- OnHook()
    - Put the active call on hook
- OffHook()
    - Put the active call off hook
- HookToggle()
    - Toggle the active call between on and off hook
- Resume()
    - Resume the active call
- Reject()
    - Reject the active call
- Accept()
    - Accept the active call
- Leave()
    - Leave the active call
- Hold()
    - Place the active call on hold
- KeypadPress(Digit)
    - Send a single digit press to the device
    - Value = string
    - Only digits (0-9) and */#/+ allowed
    - Module will also build a keypad string from digits passed in which is used with the KeypadDial() command.

- KeypadRelease(Digit)
    - Send a single digit release to the device
    - Value = string
    - Only digits (0-9) and */#/+ allowed
- KeypadClear()
    - Clear the internally stored keypad text
- KeypadBackspace()
    - Remove a character from the internally stored keypad text
- DialString(Number)
    - Set the internal keypad dial text to a specific string
    - Value = string
    - Only digits (0-9) and */#/+ allowed
- KeypadDial()
    - Dial the currently stored keypad string
- ResetList()
    - Reset the contacts list to the top level
- ResetFilterSearch()
    - Reset the filter/search field
- SetFilterSearchText(Filter)
    - Filter the current known names from the contacts list. Duplicates will not be shown.
    - Value = string
- PerformSearch()
    - Perform a deep search of all contacts using the current filter text
- SelectListItem(Index)
    - Select an item from the contact list
    - Value = integer
- DialContact()
    - Dial the selected contact from the list
- AddContactActiveSession()
    - Add the selected contact to the active call/session
- AddContactToGroup()
    - Add the selected contact to a new group. The groups to assign to will be displayed automatically. Select one of the groups to add the contact to it.
- RemoveContactFromGroup()
    - Remove the selected contact from the current group

- RemoveContactFromAllGroups()
  - Remove the selected contact from all groups

- CreateSessionUsingSelection()
  - Create a session/meeting/call. If a group is selected, all contacts in that group will be invited.

- DeleteGroup()
  - Delete the currently selected group

- MeetNow()
  - Start a new meeting now

- JoinMeeting(Url)
  - Join a meeting at the given URL
  - Value = string

- JoinConference(Url)
  - Join a conference at the given URL
  - Value = string

- SelectSession(Index)
  - Choose the Active Session/Call/Meeting from the list
  - Value = integer

- CreateGroup(Group)
  - Create a new group with the given name
  - Value = string

Direct Feedback (Request)

- IsInitialized()
  - Returns value indicating if the component is initialized (useful for debugging)
  - Return = integer (0 = Not Initialized | 1 = Initialized)

- IsRegistered()
  - Returns value indicating if the dialing component is registered (useful for debugging)
  - Return = integer (0 = Not Registered | 1 = Registered)

- HasActiveCall()
  - Returns value indicating if a call is currently active
  - Return = integer (0 - 5)

Event Feedback (Unsolicited)

- OnKeypadTextChange (Interface, Value)
  - Provides the current keypad string stored in the module
  - Interface = the object that triggered the event
  - Value = string

- OnFilterSearchTextChange (Interface, Value)
  - Provides the current filter string stored in the module
  - Interface = the object that triggered the event
  - Value = string

- OnContactListUpdate (Interface, Value)
  - Provides the current Contact List information
  - Interface = the object that triggered the event
  - Value = an 'S4BContactListResponse' object which consists of:
    - List (list of strings)
    - Type (S4BListType - integer)
    - Function (S4BListFunction - integer)
    - HelpText (string)

- OnSelectedItemChange (Interface, Value)
  - Provides the current Selected Item information
  - Interface = the object that triggered the event
  - Value = an 'S4BSelectedItem' object which consists of:
    - Type (S4BListType - integer)
    - DisplayName (string)
    - Group (string)

- OnSessionListUpdate (Interface, Value)
  - Provides the current Session List information
  - Interface = the object that triggered the event
  - Value = an 'S4BSessionListResponse' object which consists of:
    - List – list of 'S4BSession' objects, each of which consist of:
      - Conference (boolean)
      - ID (string)
      - Status (S4BSessionStatus - integer)
      - ContactUrl (string)
      - DisplayName (string)
      - StatusValue (S4BSessionStatus - integer)
      - StatusText (string)
    - ActiveItem (integer)

- OnActiveSessionInfoUpdate (Interface, Value)
    - Provides the current Active Session information
    - Interface = the object that triggered the event
    - Value = an 'S4BActiveSessionInfo' object which consists of:
        - MeetingID (string)
        - Url (string)
        - Status (S4BSessionStatus - integer)
        - IsConference (boolean)
        - StatusText (string)

- OnErrorNotification (Interface, Value)
    - Provides the current Error information
    - Interface = the object that triggered the event
    - Value = string

- OnRegistrationStatusChange (Interface, Value)
    - Provides the current Registration information
    - Interface = the object that triggered the event
    - Value = a 'S4BRegistrationStatus' object which consists of:
        - State (S4BRegState - integer)
        - Status (S4BRegState - integer)
        - StatusText (string)

Dictionaries

The following dictionaries are available for your reference. You may use these dictionaries in a read-only manner to look up keys/values in your program. Do not change any keys or values in the dictionaries as this may result in undesired operation and behavior.

```
SignalTypes = {
   'Digital': 0,
   'Analog': 1,
   'Serial': 2,
   'Null': 3
}

RampTypes = {
   'Stop': 0,
   'Raise': 1,
   'Lower': 2
}

GainTypes = {
   'GAIN': 0,
   'GAIN_FINE': 1,
   'Null': 2
}

ResponseMessageType = {
   'STACK': 0,
   'BOX': 1,
   'ROOM': 2,
   'EP': 3,
   'MT': 4,
   'MCCF': 5,
   'PROMPT_ONLY': 6,
   'UNKNOWN': 7
}

CrosspointTypes = {
   'None': 0,
   'Crosspoint': 1,
   'Gated': 3,
   'NonGated': 4,
   'PreAec': 5,
   'Unknown': 255
}

ModeTypes = {
   'Normal': 0,
   'GPIO': 1
}
```

```
EndpointType = {
    'MIC': 0,
    'TELCO_RX': 3,
    'TELCO_TX': 4,
    'VOIP_RX': 5,
    'VOIP_TX': 6,
    'OUTPUT': 7,
    'SPEAKER': 8,
    'PROC': 9,
    'FADER': 10,
    'BFM': 13,
    'USB_RX': 14,
    'USB_TX': 15,
    'UA': 20,
    'GPIO': 21,
    'SGEN': 22,
    'SRMIC': 23,
    'DANTE_RX': 24,
    'DANTE_TX': 25
}

RoomOptions = {
    'Mode': 1,
    'WallState': 2,
    'Select': 7,
    'PresetState': 9,
    'IndividualDividerState': 11,
    'DividerPolarity': 12,
    'DividerState': 13,
    'Unknown': 255
}

NotificationIndicationTypes = {
    'HOLD': 0,
    'MUTE': 1,
    'PARTY_LINE': 2,
    'RINGBACK': 3,
    'RINGING': 4,
    'CALL_WAITING_CALLEE': 5,
    'BUSY': 6,
    'WARNING_ERR': 7
}

CallForwardWaitingStatus = {
    'DISABLED': 0,
    'UNCONDITIONAL': 1,
    'BUSY': 2,
    'NO_REPLY': 3
}
```

```
StateChangeTypes = {
    'UNKNOWN': 0,
    'IDLE': 1,
    'DIAL_TONE': 2,
    'DIALING': 3,
    'INPROCESS': 4,
    'RINGING': 5,
    'BUSY': 6,
    'ACTIVE': 7,
    'HOLD': 8,
    'INCOMING': 9,
    'CFW': 10,
    'CONFERENCE_ACTIVE': 11,
    'CONFERENCE_HOLD': 12,
    'TRANSFER_ACTIVE': 13,
    'TRANSFER_HOLD': 14,
    'TRANSFERRING_DIAL_TONE': 15,
    'TRANSFERRING_DIALING': 16,
    'TRANSFERRING_INPROCESS': 17,
    'TRANSFERRING_RINGING': 18,
    'TRANSFERRING_BUSY': 19,
    'TRANSFERRING_ACTIVE': 20,
    'TRANSFERRING_HOLD': 21,
    'BLIND_TRANSFER_HOLD': 22,
    'BLIND_TRANSFERRING_DIAL_TONE': 23,
    'BLIND_TRANSFERRING_DIALING': 24,
    'BLIND_TRANSFERRING_INPROCESS': 25,
    'BLIND_TRANSFERRING_RINGING': 26
}

NotificationIndicationStatusTypes = {
    'OFF': 0,
    'ON': 1,
    'BLINK': 2
}

CallForwardWaitingStates = {
    'INACTIVE': 0,
    'INPROCESS': 1,
    'ACTIVE': 2
}

DNDStatus = {
    'DND_NOT_SET': 0,
    'DND_CALL_MUTE': 1,
    'DND_CALL_REJECT': 2
}
```

```
S4BListType = {
   'UNKNOWN': 0,
   'GROUPS': 1,
   'CONTACTS': 2,
   'INFO_ONLY': 3
}

S4BListFunction = {
   'UNKNOWN': 0,
   'SELECT_GROUP': 1,
   'SELECT_CONTACT': 2,
   'ADD_TO_GROUP': 3,
   'SEARCHING': 4
}

S4BRegState = {
   'UNKNOWN': 0,
   'REGISTERED': 1,
   'NOT_REGISTERED': 2,
   'NO_PROXY_DEFINED': 3
}

S4BSessionStatus = {
   'UNKNOWN': 0,
   'IDLE': 1,
   'CONNECTING': 2,
   'RINGING': 3,
   'BUSY': 4,
   'ACTIVE': 5,
   'HOLD': 6,
   'INCOMING': 7,
   'CONFERENCE_JOIN': 8,
   'INVITE_JOIN_AUDIO': 9
}
```

Revision History

Date	Initials	Comments
05-15-2017	DG	v1.0.0 Initial Release (Control Concepts, Inc.)
09-05-2018	DG	v1.1.0 Add Skype Component (Control Concepts, Inc.)