

IX Series multichannel networked DSP power amplifier Crestron Control API



Release Notes

Release Date	Version	Changes
08/2025	V1.0.0	<ul style="list-style-type: none">• First release.• Developed using Visual Studio 2022 and targets .NET6. The API allows you to discover and control IX Series amplifiers using Crestron 4-Series hardware and VC-4 virtual control server.• The web UI XPanel for the demo project was built using Crestron Construct v2.401.18.0 and CH5 Version 2.13.0• Requires SONICUE v1.5 or greater for initial configuration of the IX amplifier.

1. Getting started – Running the demo project

The IX Series API package folder contains the following components;

- **Dynacord.IXCrestronUI** - A Visual Studio 2022 solution folder. Double click **Dynacord IX CrestronUI Demo.sln** in the root folder to open the solution with Visual Studio (the free Community edition is supported)
- **Construct Project** – folder containing the Construct solution for the web XPanel built for the demo project. The demo UI project is contract enabled allowing the C# code in the demo solution to access UI controls by name rather than the more traditional join numbers.
- **lib** – folder containing the IX Series API dll files required for remote control. This folder can be copy/pasted to your other Crestron C# control projects when IX Series remote control will be needed. After copying, add a reference to each dll in your projects Dependencies.
- This user guide.

Step 1 - Prepare the solution to deploy on your Crestron control system:

1. Open the Dynacord IX CrestronUI Demo solution in Visual Studio. When it is first opened, the solution will need to download the Crestron SDK package from NuGet so an internet connection will be required. Visual Studio will also prompt you to install .NET6 if it isn't already installed.
2. From the Solution Explorer pane, open ControlSystem.cs
3. Scroll down to the **private async Task Init()** method and change the amplifier model in the line of code creating a new instance of IXController **IX = new IXController(Model.IX608)** to match your IX amplifier type.
4. Just below, in the IXController constructor, change the HostName property to match the host name of your amplifier. The host name can be found on the information sticker on the back panel of the device and takes the form IX-<MAC Address>, where <MAC Address> is the last 6 characters of the MAC address of the IX device. You can also find the host name from SONICUE when it is online to the amplifier. Open the OnlineMapping flyout and get the host name from the 'ServiceName' box.
5. Change the Solution Configuration from Debug to Release and build the solution.

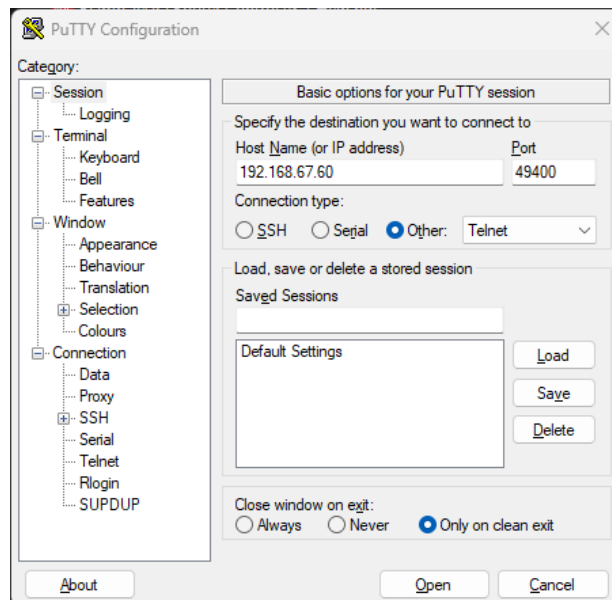
Step 2 - To install the demo project on 4-Series hardware

1. If necessary use Crestron Toolbox to enable the Web Server on the controller. Log in to the controller through Device Discovery Tool. Select 'Ethernet Settings'. In the 'Ethernet Addressing' window that appears select the 'Ethernet Ports' tab. Confirm the Web Server is enabled. The web server is required to access the controller web interface, and also to view the demo project web XPanel UI in a web browser.

Step 3 - To install the demo project on VC-4

1. Discovery and logging will require two ports to be allowed on the VC-4 firewall. These ports will be disabled by default. Assuming you followed the Crestron guidance to install VC-4 and you also installed Cockpit, log in to Cockpit and select 'Networking' from the sidebar menu.
2. On the network page, Firewall section, confirm the firewall is enabled and click 'Edit rules and zones'.
3. On the Firewall page, click 'Add services'.
4. Add UDP service on port 5353 for mdns discovery (shortcut -> enter 5353 in the 'Filter services' box, then check the mdns discovery service).
5. Add custom TCP port 49400 for VC-4 Virtual Console. This will allow you to view logging messages using a terminal service such as PuTTY.
6. Log on to the VC-4 web interface and select the 'Settings' tab.
7. In 'Program Library', click 'Add Program'
8. In the 'Add Program' window, give the program a name, such as IX Demo, then click 'Choose'
9. In the file explorer 'Open' window, browse to the location of the demo Visual Studio solution folder, inside the path to the cpz file to upload to the controller will be
[Dynacord IX CrestronUI Demo V1.0.0\Dynacord.IXCrestronUI\bin\Release\net6.0\](#)
[Dynacord.IXCrestronUI.cpz](#)
10. Select the cpz file, then click 'Open'. In the 'Add Program' window, click 'Next'.
11. In 'Add Program' Step 2, click the XPanel(Web) 'Choose' button.
12. In the file explorer 'Open' window, browse to the location of the demo Visual Studio solution folder, inside the path to the ch5z file (the web XPanel file) to upload to VC-4 will be
[Dynacord IX CrestronUI Demo V1.0.0\Construct](#)
[Project\IXCrestronUI\IXCrestronUI\output\IXCrestronUI.ch5z](#)
13. Select the ch5z file, then click 'Open'. In the 'Add Program' window, click 'Upload'.
14. Click 'Add' to add the program and close the 'Add Program' window.
15. Back on the browser main page, select the 'Status' tab > 'Rooms', then click 'Add Room'
16. In the 'Add Room' window, select the Program name you just added from the drop down and enter the Room Name and Room ID (e.g. Room1). Then click the 'Add' button.
17. Once the room has started, click your room name link in the 'Room' column of the table. Then, on the next page click the 'XPanel URL' link to open the demo project web UI in your browser.

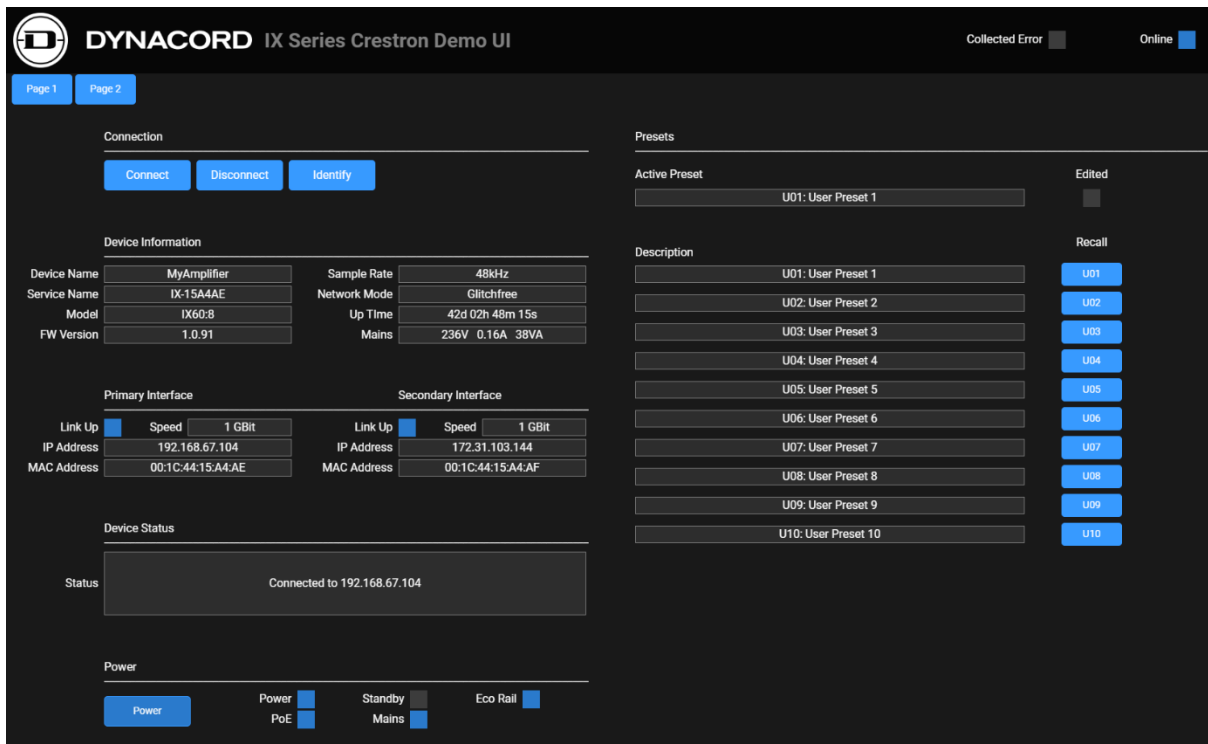
18. If the web UI will not load, or the controls and status boxes do not appear to be working, you may need to create an authentication group. On the VC-4 web interface, select the 'Settings' tab for the 'Server' (not the current room), then expand 'Authentication Management'. If there is no authentication group, click 'Add Group' then add a group with the user name you configured when installing your VC-4 instance and set the Access Level to Administrator.
19. VC-4 does not output to the Toolbox Text Console but you can still view debug logging messages using a telnet client such as PuTTY. Connect the telnet client to your VC-4 using the VC-4's IP address, the TCP port 49400, and the Telnet connection type. You MUST also enable TCP port 49400 on the VC-4 host operating systems firewall settings, otherwise logging will not work.
20. The connection to the virtual console is not authenticated and not secure. Although the host operating system cannot be accessed through the virtual console, if security is paramount for your project, you should remove the logging port from the host firewall settings before handing the system over to your customer.



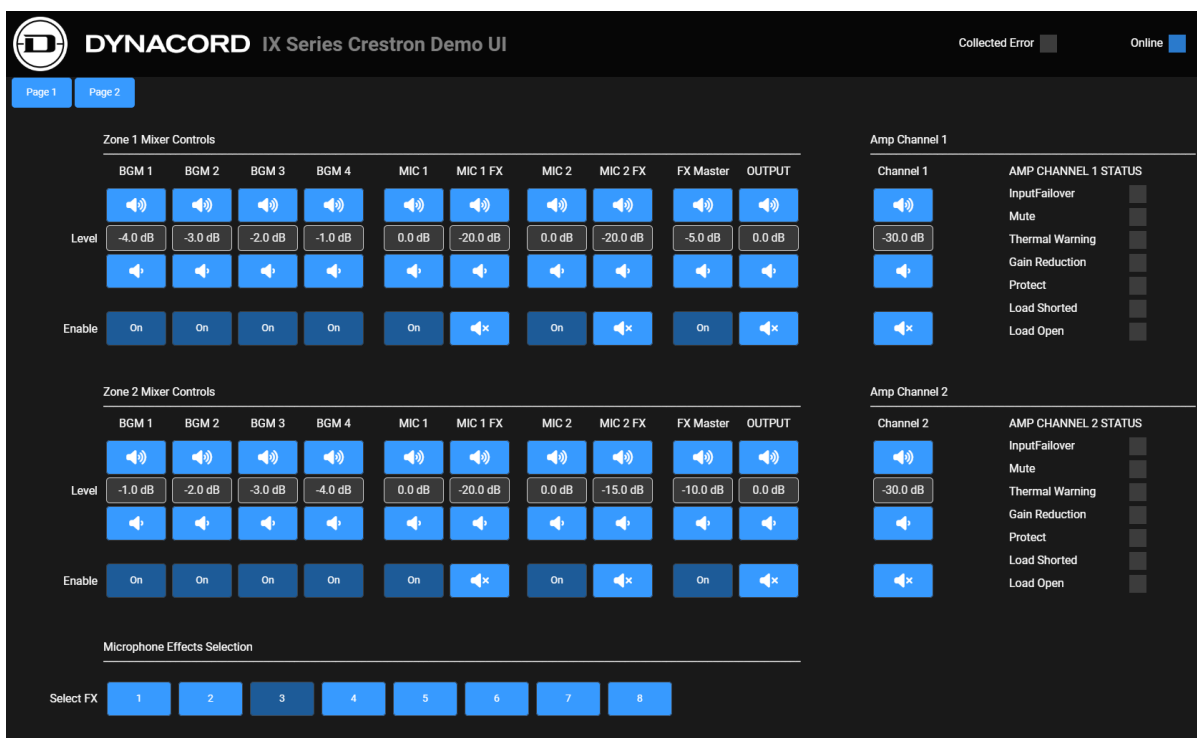
```
192.168.67.123 - PuTTY
ROOM1>
ROOM1> 07:54:40.345843 DEBUG : IXController: Sending command: {"method":"setIsFindActive","id":20618,"params":{"findActive":true}}
ROOM1> 07:54:42.066083 DEBUG : IXController: Sending command: {"method":"setIsFindActive","id":20636,"params":{"findActive":false}}
ROOM1> 07:54:44.525919 DEBUG : IXController: Sending command: {"method":"setIsPowered","id":20662,"params":{"power":false}}
ROOM1> 07:54:49.092197 DEBUG : IXController: Sending command: {"method":"setIsPowered","id":20709,"params":{"power":true}}
ROOM1> 07:54:53.033978 DEBUG : IXController: Sending command: {"method":"recallPreset","id":20749,"params":{"slot":"U03"}}
ROOM1> 07:55:16.745760 DEBUG : IXController: Polling stopped
ROOM1> 07:55:16.747249 INFO : WebSocketClient: Disconnected
ROOM1> 07:55:22.124309 INFO : IXController: Attempting to connect to wss://192.168.67.104/roadielink/v1
ROOM1> 07:55:22.244678 INFO : WebSocketClient: Connected to wss://192.168.67.104/roadielink/v1
ROOM1> 07:55:22.936794 DEBUG : IXController: Polling started with interval 100ms
ROOM1>
```

2. Demo project overview

Provided your IX amplifier is powered up and connected to the network the running program will connect to the amplifier and retrieve all the current control values and settings. To verify the you are connected, the Online LED in the top right corner of the UI will turn blue, and the Device Status box will display the 'Connected to <Your IX amplifier IP address>'



Page 2 of the UI interacts with some of the more common controls you are likely to need in a typical Creston control system.



The C# code in the Visual Studio solution shows you how to write a program integrating many aspects of the IX Series API including:

- Initializing DNS-SD discovery and starting the service to find IX devices on your network.
- Initializing an IXController instance, which is the gateway to the IX API.
- Using contracts generated in the Crestron Construct solution to access UI controls by name, rather than the more traditional join numbers.
- Logging debug messages to the 4-Series and VC-4 terminal windows.
- Registering user interaction events from the UI and how to get and set control values on the API when a UI button is pressed.
- Registering for value changed events on the API and passing those values back to the UI.

There are extensive comments in the solution class files so we recommend reviewing the solution to make yourself familiar with many of the basic concepts for working with the API in your own Crestron projects.

By all means, experiment with the code and try things out. Just remember if you make a change, rebuild the complete project, then upload it to your 4-Series or VC-4 again to test it.

3. Using the IX Series API in your own projects

This section assumes you will be using the IX Series API with a Crestron 4-Series or VC-4 control system programmed in C#. It outlines the process for creating a control system to remotely manage and control the functions of an IX amplifier. As the API is a pure C# implementation, it can also be used in a similar way to target Windows WinForms and WPF applications.

The API targets .NET6 as that is the latest version currently supported by Crestron 4-Series controllers.

You will require

- A Dynacord IX-Series amplifier (all models are supported).
- Dynacord SONICUE software V1.5 (or later) for initial configuration of the amplifier.
- Visual Studio 2022 (the free Community edition is also fully supported).
- Crestron Construct software to create a UI.
- A Crestron 4-Series hardware controller or a configured VC-4 instance.
- A good understanding of each of these systems.
- A good understanding of the C# programming language.

The IX Series API package consists of four assemblies exposed as DLL's. The DLL name is also the assembly name and the root namespace for that assembly.

- Dynacord.Utilites – helper classes used by the API, including a simple logging framework to write log messages to various output mediums. The logging namespace

Dynacord.Utilities.Logging can be referenced in your code by adding a using statement to the namespace of any classes that require it.

- Dynacord.Transport – low level transport classes for UDP and WebSocket clients. These classes are used by other assemblies in the API but don't need to be accessed directly within your program.
- Dynacord.Discovery – a DNS-SD library to discover IX devices.
- Dynacord.IXSeriesAPI – the API exposes a set of common methods to get/set parameters and subscribe to property value changed events.

The reusable API package is contained in the **Dynacord IX CrestronUI Demo** in the '**lib**' folder. This folder contains the four DLL's mentioned above, and their accompanying xml file. The xml files contain the intellisense documentation for Visual Studio.

Step 1 – Create a new Visual Studio solution:

1. Create a new Visual Studio class library project that targets .NET6
2. Once the project is created, right click on the project in the Visual Studio Solution Explorer pane and select 'Manage NuGet Packages...'.
3. From the NuGet Package Manager install the **Crestron.SimplSharp.SDK.Program** and **Newtonsoft.Json** packages.
4. Save the project, then use Windows Explorer to copy/paste the 'lib' folder from the demo solution into the root folder of your new project solution.
5. Return to Visual Studio and right click on the **Dependencies** folder for your project in the Solution Explorer pane. Select 'Add Project Reference...'
6. Select the 'Browse' option in the Reference Manager window that appears, then browse to the 'lib' folder you just copied and add all four Dynacord assembly DLL's.
7. Delete the default Class1.cs file and add a new class called ControlSystem.cs
8. You are now ready to start programming your Crestron control system.

Step 2 – Configure the control system

1. Add these namespaces to ControlSystem.cs (in addition to any other namespaces that you may require).

```
using Crestron.SimplSharp;  
using Crestron.SimplSharpPro;  
using Dynacord.Discovery.MDNS;  
using Dynacord.IXSeriesAPI;  
using Dynacord.Utilities;  
using Dynacord.Utilities.Logging;
```

2. Let ControlSystem.cs inherit from CrestronControlSystem and add a property for your IX amplifier.

```
public class ControlSystem : CrestronControlSystem
{
    public static IXController? IX { get; private set; }
}
```

3. Override the CrestronControlSystem base class **InitializeSystem()** method to call the async **Init()** method shown below.

```
public override void InitializeSystem()
{
    base.InitializeSystem();
    Init().SafeFireAndForget((Exception ex) => // do something such as logging the exception here );
}

private async Task Init()
{
    // Allow time for the control system to initialize
    // Adjust the delay time to suite the complexity of your control system initialization.
    await Task.Delay(5000);
    try
    {
        DiscoveryManager.Init();
        IX = new IXController(Model.IX608)
        {
            AutoConnect = true,

            //Remember to change the HostName to match your own IX device!
            HostName = "IX-15A4AE",
        };
        // Start the discovery service to find IX devices on the network
        DiscoveryManager.StartDiscovery("_roadielink._tcp.local");
    }
    catch (Exception ex)
    {
        // do something such as logging the exception here
    }
}
```

4. You may notice that this is the same code as used in the demo project solution. We highly recommend you use that code as the template for your own ControlSystem.cs class and simply extend it with your own methods as required. This is especially useful if you want to enable logging in your own application. You can add the logging code from the ControlSystem() constructor and copy/paste the ‘Virtual Console’ and ‘Crestron Logger’ classes from the **Logging** folder to your own solution.

Step 3 – Add methods and event handlers to control and monitor the status of the IX device

1. The IX Series API exposes all get/set methods and value changed events for the properties available on the IX device in a consistent way.
2. Again, we recommend you review the code in the ‘User Interface’ and ‘Widgets’ folders for examples of setting values and subscribing to API events.
3. The code snippets below are examples for how to change the volume level of an amplifier channel, and be notified when the volume changes from another source.

```

public class MyClass
{
    private IXController ix = ControlSystem.IX;

    public MyClass()
    {
        // Example of how to register a value changed event with an inline Lambda
        for (int i = 0; i < ix.NumAmpChannels; i++)
        {
            var idx = i // capture i for closure
            ix.Device.Channel[idx].UserProcessing.Level.ValueChanged += (s, e) =>
            {
                // idx will be the (zero based) index of the amp channel who's volume Level changed.
                // Do something with 'e.Value' which will be the new value the property changed to.
            }
        }

        // Example of how to register a value changed event with a conventional method
        ix.Device.Channel[0].UserProcessing.Level.ValueChanged += OnAmpChannel1VolumeChanged
    }

    // Example of how to get a property value
    // Property values are only accurate whilst the API is connected to the IX device.
    // All property values are stored in an in-memory database, synchronized during the
    // initial connection and maintained for as long as the connection is established.
    // Calling GetValue() on a property before this initial connection will return the
    // default value for the type.
    // Calling GetValue() on a property once the initial connection has been made will
    // return the value stored in the database, as mentioned this will be accurate so
    // long as the connection is still established.
    // GetValue() does not contact the IX device, so it does not need to be awaited, as
    // it will return the property value from memory immediately.
    public float GetAmpChannelVolume(int channel)
    {
        return ix.Device.Channel[channel].UserProcessing.Level.GetValue();
    }

    // Example of how to set a property on the remote IX device
    // ALL SetValue methods are awaited as there will be a small delay while the new value
    // is sent to the IX device over the network, and we don't want to block the app while
    // we wait.
    // Always wrap async/await methods in a try/catch block (or use a safe fire and forget
    // wrapper around the method call) to catch any exceptions that may occur during the
    // awaited operation.
    public async Task SetAmpChannelVolume(int channel, float newVolumeLevel)
    {
        try
        {
            await ix.Device.Channel[channel].UserProcessing.Level.SetValue(newVolumeLevel);

            // If you are not concerned which thread the awaited operation returns on you can
            // chain the above call with .ConfigureAwait(false) like this;
            // await
            // ix.Device.Channel[channel].UserProcessing.Level.SetValue(newVolumeLevel).ConfigureAwait(false);
            // to avoid an unnecessary context switch.
        }
        catch (Exception ex)
        {
            // handle any exceptions from the async/await method call
        }
    }

    // This method will be invoked whenever the volume level for amp channel 1 changes
    private void OnAmpChannel1VolumeChanged(object? sender, EventArgs<float> e)
    {
        // Do something with 'e.Value' which will be the new value the property changed to.
    }
}

```

4. Refer to the Crestron documentation in the Crestron NuGet package for further guidance on working with the CrestronControlSystem classes and SimplSharp/SimplSharpPro namespaces.

4. Navigating the API

To use the API requires instantiation of just one class (**IXController**), with a separate instance required for each IX device you need to control. **IXController** exposes methods to handle connection to the device, in particular the **Device** property which represents the entry point into all controllable properties on the device. Property access is simply chained from **Device**. For example, assuming you created an instance field of **IXController** named **ix**

```
private IXController ix = new(Model.IX608);
```

all device properties will be accessed by chaining to the required properties like this;

```
public void MyMethod()
{
    string ampName = ix.Device.Name.GetValue();
    float myLevel = ix.Device.Mixer.Crosspoint[2, 5].Level.GetValue();
    bool amp1Mute = ix.Device.Channel[0].UserProcessing.Mute.GetValue();
}
```

All controllable properties expose a common interface allowing you to call two methods **GetValue()** and **SetValue(T newValue)**, and subscribe to a single **ValueChanged** event.

For every device property **SetValue(T newValue)** is an async method so it is highly recommended that your call is wrapped in a try/catch block (or a safe fire and forget extension method included in the **Dynacord.Utilities** namespace – see the demo project for example usage) to catch any exceptions raised by the async/await state machine.

```
public async Task BypassCompressor(bool bypass)
{
    try
    {
        await ix.Device.NetInput[2].Compressor.Bypass.SetValue(bypass);
    }
    catch (Exception ex)
    {
        // handle or log any exceptions raised by async/await here
        // to prevent your app from crashing
    }
}
```

The API is quite extensive, exposing several thousand individual properties on the IX device. The following section describes some of the most likely device properties you will want to control with your Crestron control system. However, it is not exhaustive, but by using Visual Studio intellisense and the intellisense code comments included in the API, you should find navigation to the less common properties not mentioned below will still be quite straightforward. The main thing to remember is all properties implement the same interface to get, set and subscribe to value changed events in the same way.

Class IXController

This class is the entry point into the API, create an instance of the class (passing in your IX amplifier type) for each IX device you want to control.

Constructor

```
public IXController(Model model)
```

Properties

ActiveIpAddress	<p>Gets the active IP address associated with the current connection. If using the HostName for device discovery this will be the resolved IP address obtained through mDNS. If using a static IP address, this will be the same value as the StaticIpAddress property.</p>
AutoConnect	<p>Specifies whether the IXController instance should automatically attempt to connect to the IX device at startup. Connection will only commence if either the StaticIpAddress property is set, or the device has been resolved through discovery.</p> <p>The default value is false, however it is common for the control system to connect on startup without user intervention, so you may well want to set this property to true.</p>
AutoConnectDelay	<p>Specifies the delay (in milliseconds) before the IXController will attempt to automatically connect to the IX device after control system startup.</p> <p>The default value is 3000 milliseconds (3 seconds). The allowed range is between 500 milliseconds (0.5 seconds) and 60000 milliseconds (1 minute). Values outside of this range will be clamped to the nearest limit. This property is only used if AutoConnect is true.</p> <p>The delay time chosen should allow sufficient time for the control system to be fully initialized.</p> <p>The AutoConnectDelay property is primarily used when a static IP address is set. If the device is being discovered via its host name, the controller will automatically connect as soon as it is discovered. If the device is not available when the delay expires, auto connect will then still be attempted to inform calling code that the connection has been initiated.</p>
AutoReconnect	<p>Specifies whether the IXController instance should automatically attempt to reconnect to the IX device after a connection is lost.</p> <p>The default value is true.</p>
AutoReconnectInterval	<p>Specifies the interval, in seconds, between automatic reconnection attempts. This property is only used if AutoReconnect is true.</p> <p>The default value is 5 seconds. The allowed range is between 1 second and 3600 seconds (1 hour). Values outside of this range will be clamped to the nearest limit.</p>

ChangeSetPollInterval	<p>Specifies the interval (in milliseconds) for change set polling calls. Change set polling calls are used to keep the API in sync with the IX device.</p> <p>The default value is 100ms. The allowed range is between 50ms and 10,000ms (10 seconds). Values outside of this range will be clamped to the nearest limit. For the UI to remain responsive to property changes on the device, the default value is optimal and values greater than 1 second should generally be avoided.</p>
Device	<p>Provides access to the DSP processing blocks available on the device.</p> <p>See the 'Device' section below for more details.</p>
DeviceStatus	<p>Gets a message indicating the current state of the device. This message could be displayed in a UI element to inform the user of current status etc.</p>
HostName	<p>The host name of the remote device this controller instance is associated with.</p> <p>The host name is used to resolve the IP address of the device through mDNS discovery. If both the host name and a static IP address are specified, the static IP address will take precedence when connecting.</p> <p>The HostName can be found on the information label on the rear of the device or through the OnlineMapping flyout in SONICUE. It takes the form IX-XXXXXX, where XXXXXX is the last six digits of the device's MAC address.</p>
IsConnected	<p>Indicates whether the IXController instance is currently connected to an IX device. When false, and SessionActive is true, the client has lost communications with the device.</p>
IsSessionActive	<p>Indicates whether the IXController instance session is currently active. When true, the client is connected to, attempting to connect, or attempting to reestablish a connection with, an IX device.</p>
Model	<p>Gets the model of the IX amplifier this IXController instance is associated with. This is the model you assigned in the IXController constructor.</p>
NumAmpChannels	<p>Gets the number of amplifier channels available on the device. This will be four or eight, depending on the model.</p>
StaticIpAddress	<p>The static IP address of the remote device this IXController instance is associated with.</p> <p>The static IP address will take precedence over the HostName when connecting. A static IP address must first be assigned to the physical device using the Network flyout in SONICUE. It can be used when DHCP and/or multicast DNS-SD discovery is not desirable for your network but be aware that any change to the device IP address will require your Crestron program to be updated with that new address.</p> <p>If you do not want to use a static IP address leave this property as an empty string.</p>

Methods

ConnectAsync()	<p>Asynchronously connects to the IX device. Either the StaticIpAddress or HostName property must be set before calling this method. If both properties are set, the StaticIpAddress will take precedence.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p> <p>An example of using this method is given in the demo project – see UserInterface > Page1.cs</p>
DisconnectAsync()	<p>Asynchronously disconnects from the remote device.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p> <p>An example of using this method is given in the demo project – see UserInterface > Page1.cs</p>
Reboot()	<p>Asynchronously sends a command to reboot the IX device. Once called, the connection will be lost and the device unavailable until its reboot cycle is complete - this may take a few minutes.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
SetLoggingLevel(LogLevel)	<p>Sets the logging level for this class instance.</p> <p>For messages to be logged you must also provide a suitable Crestron logger to the 'LogDispatcher' static class. The demo project includes a set of classes (in the 'Logging' folder) to log to Crestron 4-Series and VC-4. You can use these classes in your own projects to add Crestron logging capability.</p> <p>For example, to add the Crestron logger for 4-Series to the LogDispatcher call</p> <pre>LogDispatcher.AddLogger(CrestronLogger.GetLogger());</pre> <p>close to the entry point in your application.</p>

Events

DeviceStatusChanged	<p>Event raised when the general status of the device changes.</p> <p>The EventArgs 'Value' property for this event provides a string representation of current status, which may contain messages about network and connectivity issues active on the device. Its primary purpose is to display status information to the user via a suitable UI control.</p> <p>An example of using this event is given in the demo project – see UserInterface > Page1.cs</p>
IsConnectedChanged	<p>Event raised when the connection state changes.</p> <p>If the EventArgs 'Value' property is true, the IXController instance is successfully connected to the IX device. If false (and the IsSessionActive property is true), the IXController instance has lost communications with</p>

	<p>the device. If false (and the <code>IsSessionActive</code> property is also false), the <code>IXController</code> instance has been gracefully disconnected.</p> <p>An example of using this event is given in the demo project – see <code>UserInterface > Page1.cs</code></p>
--	---

IXController.Device

All DSP and control objects on the API can be accessed through this property. See the introduction to Navigating the API for examples of chaining to the required properties.

Properties

Channel[]	Gets the (zero based) array of amplifier channel DSP processing blocks for this device. The number of channels matches the number of amplifier channels on the physical IX device. Channel one is at index 0, channel two at index 1 and so on.
ChannelPilotToneFrequency	Gets the load supervision pilot tone frequency property for this device.
ControlPort	Gets the control port properties for this device.
Fx	Gets the effects processing engine for this device.
Id	Gets the device id property for this device.
Identify	<p>Gets the identify property for this device.</p> <p>Setting <code>Identify</code> to true causes the front panel indicator to flash enabling the physical device to be located in a rack.</p>
MicInput	Gets the (zero based) array of microphone input processing groups for this device. Four channel IX amplifiers have four microphone inputs available, eight channel amplifiers have eight microphone inputs.
Mixer	Gets the matrix mixer DSP block for this device.
Model	<p>Gets the model type property for this device. This is the model type returned by the IX amplifier. It is used to verify the user assigned model in the <code>IXController</code> constructor actually matches that of the device.</p> <p>Any connection to a wrong model type will be aborted.</p>
MuteByRelay	Gets the mute by relay property for this device.
Name	<p>Gets the device name property for this device.</p> <p>The name is the device label applied in SONICUE and is also the name used to discover the IX amplifier in Dante Controller.</p>

NetInput	Gets the (zero based) array of network audio input processing groups for this device. All IX models have eight Dante networked audio inputs.
NetOutput	Gets the (zero based) array of network output processing groups for this device. All IX models have eight Dante networked audio outputs.
NetOutputPilotToneFrequency	Gets the network output pilot tone frequency property for this device. The pilot tone frequency is common to all Dante network outputs on the device. When enabled on a network output channel, the pilot tone sinewave is superimposed on to the Dante audio signal to allow a downstream device to monitor the audio integrity of the network output.
Network	Gets the network configuration properties for this device.
OnTime	Gets the ontime property (in seconds) for this device. OnTime is the total number of seconds the device has been powered up since it was manufactured.
PowerActualState	Gets the power state property for this device.
PowerConsumption	Gets the power consumption properties for this device. Power consumption reports the actual mains voltage and current being consumed by the amplifier power supply.
PowerTarget	Gets the power target property for this device. Setting the power target to true/false will switch the amplifier between powered on and standby.
Presets	Gets the presets property for this device. All IX devices provide 20 user presets and one factory preset.
SampleRate	Gets the sample rate property for this device. All IX devices support 48 kHz and 96 kHz sample rates.
ServiceName	Gets the service name property for this device. Also known as the host name and takes the form 'IX-XXXXXX', where 'XXXXXX' is the last six characters of the device's MAC address.
SignalGenerator	Gets the signal generator properties for this device.
Status	Gets the device state flags for this device. These are state flags specific to the device itself, rather than channels or input sources. Each of the later have their own groups of state flags per channel or input.
TaskEngine	Gets the task engine 'virtual' objects for this device.
Temperature	Gets the temperature properties for this device.
Time	Gets the time properties for this device.
Version	Gets the firmware version running on the device.

Methods

SetSampleRate(SampleRates)	<p>Asynchronously sets the sample rate on the IX device. Note: Changing the sample rate will cause the device to reboot. This process may take a few minutes.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
----------------------------	--

IXController.Device.Channel[]

Access to the array of channel based properties. You must specify the (zero based) index of the channel you want to access -> IXController.Device.Channel[0] etc. The number of available channels matches the number of channels on the physical IX device (either four or eight).

Properties

AmpBridgedMode	<p>Gets the bridged mode property for this amplifier channel.</p> <p>Set to true to bridge the channel; otherwise false.</p>
AmpDriveMode	<p>Gets the drive mode property for this amplifier channel.</p> <p>The amplifier output drive has several low and high impedance options defined by the AmplifierDriveMode enum.</p>
AmpGainInDb	<p>Gets the amp gain property for this amplifier channel.</p>
ArrayProcessing	<p>Gets the array processing block for this amplifier channel. These are the speaker 'Array' settings in SONICUE</p> <p>Chain from ArrayProcessing to reach the array speaker processing for</p> <ul style="list-style-type: none">• Delay• EQ• Polarity• Trim
FailoverMode	<p>Gets the failover mode property for this amplifier channel.</p> <ul style="list-style-type: none">• Failover Fallback,• Failover No Fallback,• Default Static,• Failover Static
FailoverTime	<p>Gets the failover time property for this amplifier channel.</p> <p>The number of seconds to wait before switching to the failover input source.</p>
FallbackTime	<p>Gets the fallback time property for this amplifier channel.</p>

	The number of seconds to wait before switching to back to the default input source.
InputRoute	Gets the input route property for this amplifier channel. Input route determines which default input sources are routed to this amplifier channel.
InputRouteFailover	Gets the input route failover property for this amplifier channel. Input route failover determines which input sources are routed to this amplifier channel when it switches to failover.
Name	Gets the name property for this amplifier channel.
PilotToneActive	Gets the pilot tone active property for this amplifier channel. True if the pilot generator is enabled; otherwise false.
PilotToneLevel	Gets the pilot tone level property for this amplifier channel.
SignalGeneratorEnabled	Gets the signal generator enabled property for this amplifier channel. True if the signal generator is enabled; otherwise false.
SignalGeneratorLevel	Gets the signal generator level property (in dBU) for this amplifier channel.
Status	Gets the state flags for this amplifier channel.
UserProcessing	The user processing block for this amplifier channel. These are the speaker 'User' settings in SONICUE Chain from UserProcessing to reach the user speaker processing for <ul style="list-style-type: none"> • Delay • EQ's • Level • Mute Note: Level and Mute are the master amplifier output Level and Mute controls for the specified amplifier channel.
ZMax	Gets the ZMax property for this amplifier channel. This property determines the maximum loudspeaker impedance threshold for speaker supervision.
ZMaxPilot	Gets the ZMaxPilot property for this amplifier channel. This property determines the maximum loudspeaker impedance threshold for speaker supervision using the band filtered pilot tone signal.
ZMin	Gets the ZMin property for this amplifier channel. This property determines the minimum loudspeaker impedance threshold for speaker supervision.

ZMinPilot	<p>Gets the ZMinPilot property for this amplifier channel.</p> <p>This property determines the minimum loudspeaker impedance threshold for speaker supervision using the band filtered pilot tone signal.</p>
-----------	---

Methods

SetDriveMode(AmplifierDriveMode)	<p>Command to set the drive mode property for this amplifier channel.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
SetFailoverMode(FailoverMode)	<p>Command to set the failover mode for this amplifier channel.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
SetZMax(float)	<p>Command to set the maximum load impedance threshold for this amplifier channel.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
SetZMaxPilot(float)	<p>Command to set the maximum pilot impedance threshold for this amplifier channel.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
SetZMin(float)	<p>Command to set the minimum load impedance threshold for this amplifier channel.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
SetZMinPilot(float)	<p>Command to set the minimum pilot impedance threshold for this amplifier channel.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>

IXController.Device.MicInput[]

Access to the array of microphone input based properties. You must specify the (zero based) index of the input you want to access → IXController.Device.MicInput[0] etc. The number of available inputs matches the number of channels on the physical IX device (either four or eight).

Properties

AGC	<p>Gets the AGC DSP block for this microphone input.</p> <p>Chain from AGC to access its properties.</p>
-----	--

Compressor	Gets the Compressor DSP block for this microphone input. Chain from Compressor to access its properties.
DspConfig	Gets the DSP configuration property for this microphone input. Selects between the AGC or Compressor DSP block for the input.
EQ	Gets the array of EQ DSP blocks for this microphone input. There are four EQ filters available. Chain from EQ[EqIdx] to access its properties.
FxSendLevel	Gets the FX send level property for this microphone input.
FxSendMute	Gets the FX send mute property for this microphone input.
Gain	Gets the mic gain property (in dB) for this microphone input. The gain is clamped between 0.0 and 60.0 dB
HipassFilter	Gets the HipassFilter DSP block for this microphone input. Chain from HipassFilter to access its properties.
Level	Gets the level property (in dB) for this microphone input. The allowed range is between -80.0 dB and 10.0 dB
Linked	Gets the linked property for this microphone input. Only valid for (zero based) even mic inputs (0, 2, 4, 6), will be null for odd inputs (1, 3, 5, 7).
Mute	Gets the mute property for this microphone input.
Name	Gets the name property for this microphone input.
NoiseGate	Gets the NoiseGate DSP block for this microphone input. Chain from NoiseGate to access its properties.
PhantomPower	Gets the phantom power property for this microphone input.
PilotDetection	Gets the PilotDetection DSP block for this microphone input. Chain from PilotDetection to access its properties.
Status	Gets the state flags for this microphone input.
Trim	Gets the trim property (in dB) for this microphone input. The trim is clamped between -30.0 and 12.0 dB
Vu	Gets the VU meter level for this microphone input.

Methods

SetDspConfig(DspInputProcessingConfig)	<p>Sets the DSP configuration for this microphone input, either AGC or compressor.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
--	---

IXController.Device.NetInput[]

Access to the array of network (Dante) input based properties. You must specify the (zero based) index of the input you want to access -> IXController.Device.NetInput[0] etc. There are eight network inputs available.

Properties

AGC	Gets the AGC DSP block for this network input. Chain from AGC to access its properties.
Compressor	Gets the Compressor DSP block for this network input. Chain from Compressor to access its properties.
DspConfig	Gets the DSP configuration property for this network input. Selects between the AGC or Compressor DSP block for the input.
EQ	Gets the array of EQ DSP blocks for this network input. There are four EQ filters available. Chain from EQ[EqlIdx] to access its properties.
FxSendLevel	Gets the FX send level property for this network input.
FxSendMute	Gets the FX send mute property for this network input.
HipassFilter	Gets the HipassFilter DSP block for this network input. Chain from HipassFilter to access its properties.
Level	Gets the level property (in dB) for this network input. The allowed range is between -80.0 dB and 10.0 dB
Linked	Gets the linked property for this network input. Only valid for (zero based) even net inputs (0, 2, 4, 6), will be null for odd inputs (1, 3, 5, 7).
Mute	Gets the mute property for this network input.
Name	Gets the name property for this network input.
NoiseGate	Gets the NoiseGate DSP block for this network input. Chain from NoiseGate to access its properties.
PilotDetection	Gets the PilotDetection DSP block for this network input. Chain from PilotDetection to access its properties.
Status	Gets the state flags for this network input.
Trim	Gets the trim property (in dB) for this network input. The trim is clamped between -30.0 and 12.0 dB.
Vu	Gets the VU meter level for this network input.

Methods

SetDspConfig(DspInputProcessingConfig)	<p>Sets the DSP configuration for this network input, either AGC or compressor.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
--	--

IXController.Device.NetOutput[]

Access to the array of network (Dante) output based properties. You must specify the (zero based) index of the output you want to access → IXController.Device.NetOutput[0] etc. There are eight network (Dante) outputs available.

Properties

InputRoute	<p>Gets the input route property for this network output.</p> <p>The input route determines the source signal that will be routed to this Dante channel.</p>
Name	Gets the name property for this network output.
PilotToneActive	Gets the pilot tone active property for this network output.
PilotToneLevelInDb	Gets the pilot tone level property (in dB) for this network output.

Methods

SetNetOutputInputRoute(NetOutputInputRoute)	<p>Sets the input source for this network output.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
---	--

IXController.Device.Mixer

Access to the input matrix mixer.

Properties

Crosspoint	<p>Gets the crosspoint matrix for the matrix mixer.</p> <p>Crosspoints are accessed via the (zero based) input index and output index of the required crosspoint. For example,</p> <p><code>IXController.Device.Mixer.Crosspoint[inIdx, outIdx].Level</code> <code>IXController.Device.Mixer.Crosspoint[inIdx, outIdx].Mute</code></p>
------------	--

	<p>Available matrix mixer output properties</p> <ul style="list-style-type: none"> • Level • Mute <p>Four channel IX amplifiers have 14 crosspoint inputs and 12 crosspoint outputs.</p> <p>Eight channel IX amplifiers have 18 crosspoint inputs and 16 crosspoint outputs.</p>
Output	<p>Gets the array of mixer outputs.</p> <p>Outputs are accessed via the (zero based) index of the required mixer output. For example,</p> <pre>IXController.Device.Mixer.Output[outIdx].Level IXController.Device.Mixer.Output[outIdx].Mute</pre> <p>Available matrix mixer output properties</p> <ul style="list-style-type: none"> • Level • Linked • Mute • Name • Vu <p>Four channel IX amplifiers have 12 mixer outputs.</p> <p>Eight channel IX amplifiers have 16 mixer outputs.</p>

IXController.Device.Fx

Access to the effects properties. The effects engine provides a range of different reverbs, echo's, delay's, chorus, flanger effects that can be added to any combination of inputs to the matrix mixer.

Properties

Delay	Gets the delay property for the FX processing block.
LoadedIdx	Gets the index of the effect currently selected.
Mute	Globally mutes the currently selected effect.

Methods

LoadFx(FxIdx)	<p>Loads the specified effects type into the FX processing block. FxIdx is an enum of all available effects type. Pass the enum value for the effect you want to load into the LoadFx method.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
---------------	--

IXController.Device.Presets

Access to the Presets properties

Properties

CurrentLoadedSlot	Gets the currently loaded preset slot (read only). This will be in the form “U01”, “U02”, “F01” etc.
InitialLoadedSlot	Gets the initially loaded preset slot (read only). This is the preset that will be loaded on amplifier startup and in the form “U01”, “U02”, “F01” etc.
IsCurrentPresetEdited	Gets the current preset edited property (read only). True if the preset has been edited (some control values are different to when the preset was saved); false if the preset’s control values haven’t changed.
Title(PresetSlotName)	Gets the preset title property for the specified slot.

Methods

GetTitles()	Gets a list of preset titles for all available preset slots. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
RecallPreset(PresetSlotName)	Command to recall the preset at the specified slot. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
SetInitialPreset(PresetSlotName)	Command to set the preset to be recalled on device startup. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
SetPresetTitle(PresetSlotName, string)	Command to set the title of the preset at the specified slot. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
StorePreset(PresetSlotName)	Command to store a preset at the specified slot. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.

IXController.Device.ControlPort.GPIO[]

Access to the (zero based) array of General Purpose Input/Output (GPIO) control ports. All IX amplifiers have three GPIO’s.

Properties

Analog	Gets the analog GPI reading for this GPIO (read only).
Digital	Gets the digital GPI state for this GPIO (read only).
Gpo	Gets the GPO property for this GPIO.
Mode	<p>Gets the mode property for this GPIO. The mode property determines whether the GPIO set to</p> <ul style="list-style-type: none">• Analog Input• Digital Input• Digital Output.

Methods

SetMode(GpioMode)	<p>Sets the operational mode of this GPIO.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
-------------------	---

Copyrights

SIMPL, SIMPL+, SIMPL#, Crestron Toolbox & Crestron Construct are trademarks of Crestron Electronics, Inc.
Dante is a trademark of Audinate Pty Ltd.
All other trademarks are the property of their respective owners.

Bosch Security Systems, LLC

130 Perington Parkway
Fairport, NY 14450, USA

www.dynacord.com

© Bosch Security Systems, Inc. 2025