

VZX-8 8-Zone Audio Processor

Crestron Control API



Release Notes

Release Date	Version	Changes
02/2026	v1.0.0	<ul style="list-style-type: none">• First release.• Developed using Visual Studio 2026 and targets Microsoft .NET6. Requires the NuGet package Crestron.SimplSharp.SDK.Program version 2.21.184 or later, which can be downloaded and installed using the Visual Studio NuGet Package Manager.• The API allows you to discover and control VZX-8 audio processors using Crestron 4-Series hardware and VC-4 virtual control server.• The web UI XPanel for the demo project was built using Crestron Construct v2.801.22.0 and CH5 Version 2.17.0

1. Getting started – Running the demo project

The VZX-8 API package folder contains the following components;

- **Dynacord.VZXCrestronUI** - A Visual Studio 2026 solution folder. Double click **Dynacord VZX 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 VZX-8 Series API dll files required for remote control. This folder can be copy/pasted to your other Crestron C# control projects when VZX-8 remote control will be needed. After copying, add a reference to each dll in your projects Dependencies.
- This user guide.

These instructions describe how to build and upload the project to your Crestron control system but a good working knowledge of tools such as Visual Studio, Crestron Toolbox, Crestron Construct and the C# programming language is assumed.

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

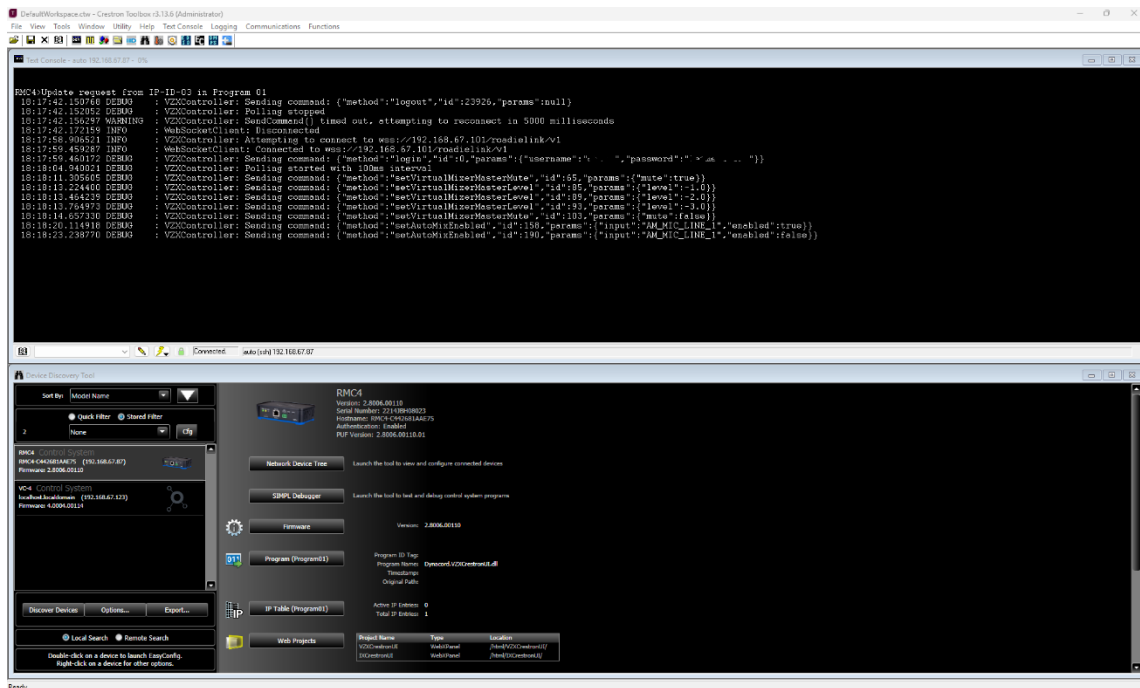
1. Open the Dynacord VZX CrestronUI Demo solution in Visual Studio. When it is first opened, the solution will need to download the Crestron SDK package from NuGet so an internet connection will be required. Visual Studio will also prompt you to install .NET6 if it isn't already installed. If you are installing the Crestron NuGet package manually you will need the **Crestron.SimplSharp.SDK.Program** package version 2.21.184 or later. This package should include the Newtonsoft.Json library, if not, also add the Newtonsoft.Json package version 13.0.4 or later.
2. From the Solution Explorer pane, open ControlSystem.cs
3. Scroll down to the **private async Task Init()** method and change the VZX 'HostName' property in the section of code creating a new instance of VZXController **VZX = new VZXController(Model.VZX8, WebSocketScheme.Secure)** to match your VZX-8 processor. The host name can be found on the information sticker on the back panel of the device and takes the form VZX-<MAC Address>, where <MAC Address> is the last 6 characters of the MAC address of the VZX-8 device.
4. In a similar way, change the 'Username' and 'Password' properties to match those assigned to your device.
5. If you want to specify a static IP address, rather than using device discovery to resolve the IP address assigned to your VZX-8, uncomment and update the 'StaticIpAddress' property accordingly.

6. There are plenty of comments in the various class files of the solution to help you better understand what the code base is doing, so taking a moment to read through them is highly recommended.
7. Change the Solution Configuration from Debug to Release and build the solution. This will create the runtime package (the Crestron cpz file) that needs to be uploaded to your Crestron controller (4-Series or VC-4).

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.
2. Log on to the controller's web interface in your Web Browser and select the 'Settings' tab.
3. In 'System Setup' ensure 'Web XPanel' is enabled.
4. In 'Programs' select the button to 'Upload Program' for the program slot you want to use.
5. In the pop up 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 VZX CrestronUI Demo V1.0.0\Dynacord.VZXCrestonUI\bin\Release\net6.0\Dynacord.VZXCrestonUI.cpz](#)
6. Select the cpz file and load it to the controller. Once registered, the program name for the Program should be Dynacord/VZXCrestonUI.dll
7. In 'Projects' click the 'Add Project' button then, in the Add Project pop up window enable the 'Web Project' switch and click the Upload button.
8. In the File Upload window click the Browse button and browse to the location of the demo Visual Studio solution folder, inside the path to the ch5z file (the web XPanel file) will be [Dynacord VZX CrestronUI Demo V1.0.0\Construct Project\VZXCrestonUI\VZXCrestonUI\output\VZXCrestonUI.ch5z](#)
9. Select the ch5z file and load it to the controller. Once complete, the project name should be VZXCrestonUI.ch5z and it should show as an active Web Project.
10. In your web browser, you can now access the web XPanel UI with the address <https://<ControllerIP>/html/VZXCrestonUI/index.html>
11. You can of course use Crestron Toolbox to load your program and UI should you prefer that method.

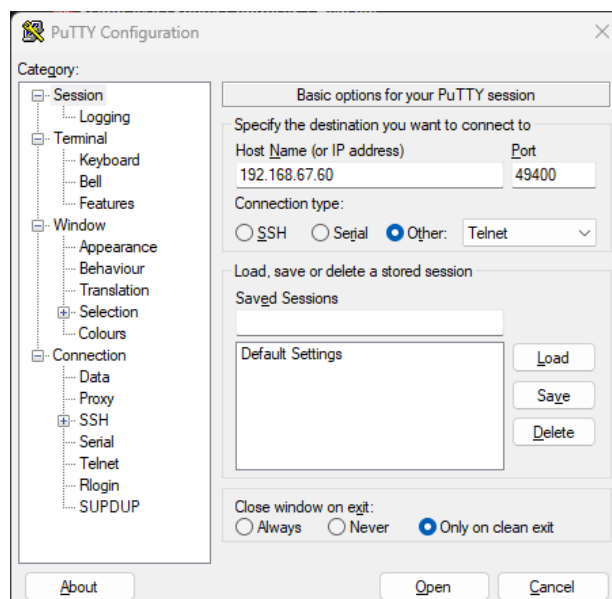
12. To view debug logging messages, log in to the controller through Crestron Toolbox Device Discovery Tool, then select 'Functions > Text Console'.



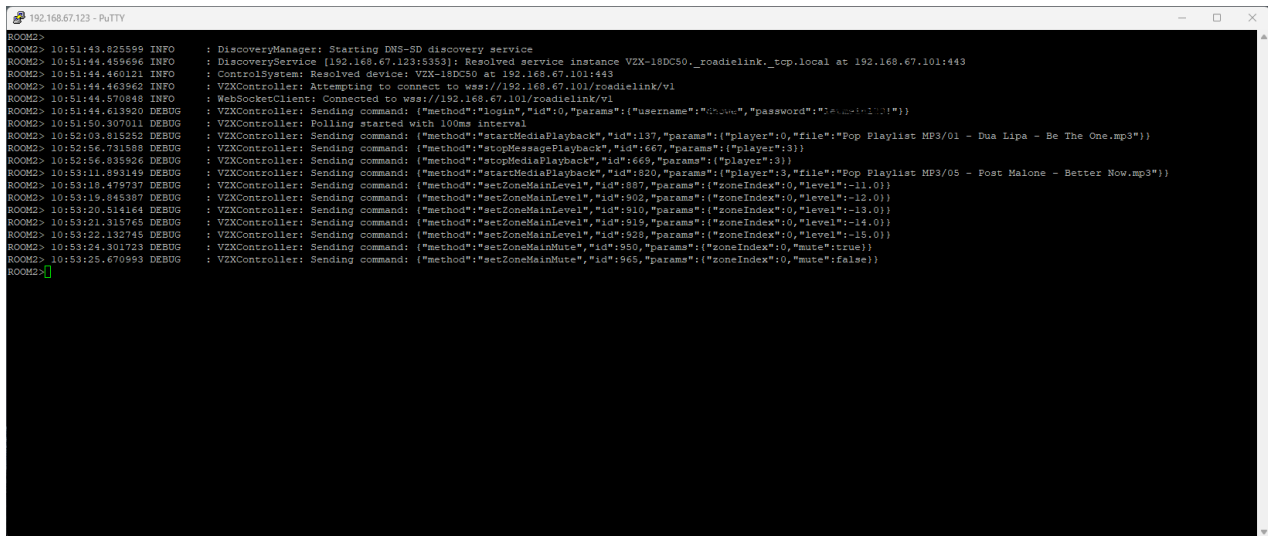
Step 3 - To install the demo project on VC-4

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 VZX 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 VZX CrestronUI Demo V1.0.0\Dynacord.VZXCrestronUI\bin\Release\net6.0\Dynacord.VZXCrestronUI.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 VZX CrestronUI Demo V1.0.0\Construct Project\VZXCrestronUI\VZXCrestronUI\output\VZXCrestronUI.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.



```
ROOM2> 10:51:43.825599 INFO : DiscoveryManager: Starting DNS-SD discovery service
ROOM2> 10:51:44.459696 INFO : DiscoveryService [192.168.67.123:5353]: Resolved service instance VZX-18DC50_roadielink_tcp.local at 192.168.67.101:443
ROOM2> 10:51:44.460121 INFO : ControlSystem: Resolved device: VZX-18DC50 at 192.168.67.101:443
ROOM2> 10:51:44.463962 INFO : VZXController: Attempting to connect to wss://192.168.67.101/roadielink/v1
ROOM2> 10:51:44.570846 INFO : WebsocketClient: Connected to wss://192.168.67.101/roadielink/v1
ROOM2> 10:51:44.613500 DEBUG : VZXController: Sending command: {"method":"login","id":0,"params":{"username":"Demo","password":"Demo1234"}}
ROOM2> 10:51:50.307011 DEBUG : VZXController: Polling started with 100ms interval
ROOM2> 10:52:03.815252 DEBUG : VZXController: Sending command: {"method":"startMediaPlayback","id":137,"params":{"player":0,"file":"Pop Playlist MP3/01 - Dua Lipa - Be The One.mp3"}}
ROOM2> 10:52:06.731588 DEBUG : VZXController: Sending command: {"method":"stopMessagePlayback","id":667,"params":{"player":3}}
ROOM2> 10:52:06.935206 DEBUG : VZXController: Sending command: {"method":"stopMediaPlayback","id":169,"params":{"player":3}}
ROOM2> 10:53:11.993149 DEBUG : VZXController: Sending command: {"method":"startMediaPlayback","id":820,"params":{"player":3,"file":"Pop Playlist MP3/05 - Post Malone - Better Now.mp3"}}
ROOM2> 10:53:18.479737 DEBUG : VZXController: Sending command: {"method":"setZoneMainLevel","id":887,"params":{"zoneIndex":0,"level":-11.0}}
ROOM2> 10:53:19.845397 DEBUG : VZXController: Sending command: {"method":"setZoneMainLevel","id":902,"params":{"zoneIndex":0,"level":-12.0}}
ROOM2> 10:53:20.814144 DEBUG : VZXController: Sending command: {"method":"setZoneMainLevel","id":910,"params":{"zoneIndex":0,"level":-13.0}}
ROOM2> 10:53:21.315765 DEBUG : VZXController: Sending command: {"method":"setZoneMainLevel","id":919,"params":{"zoneIndex":0,"level":-14.0}}
ROOM2> 10:53:22.132745 DEBUG : VZXController: Sending command: {"method":"setZoneMainLevel","id":928,"params":{"zoneIndex":0,"level":-15.0}}
ROOM2> 10:53:24.301723 DEBUG : VZXController: Sending command: {"method":"setZoneMainMute","id":950,"params":{"zoneIndex":0,"mute":true}}
ROOM2> 10:53:25.670993 DEBUG : VZXController: Sending command: {"method":"setZoneMainMute","id":965,"params":{"zoneIndex":0,"mute":false}}
ROOM2>
```

2. Demo project overview

The C# code in the Visual Studio solution shows you how to write a program integrating many aspects of the VZX-8 API including:

- Initializing DNS-SD discovery and starting the service to find VZX-8 devices on your network.
- Initializing a VZXController instance, which is the gateway to the VZX 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.

Provided your VZX-8 is powered up and connected to the network, the running demo program will auto connect to the processor and retrieve all the current control values and settings. To verify that you are connected, the Online LED in the top right corner of the UI will turn blue, and the 'Device Status' box on the Device page will display the 'Connected to <Your VZX-8 IP address>' message.

Device

The Device page gives an overview of device information and status as well as selecting the inputs that should appear on the VZX- Virtual Mixer, and controls for the built-in media players.

The connect and disconnect buttons show how manual online/offline can be configured, they are optional however for your own projects. If you want your control system to always be connected to the VZX-8, make sure the API 'AutoConnect' and 'AutoReconnect' properties are both set to true. This will ensure that the control system automatically connects to the VZX-8 at startup and will then constantly try to reconnect to the device whenever the network connection is lost. This means the control system user doesn't need to worry about connecting to the device as it will always be handled automatically.

Virtual mixer source selection controls determine which controls will be enabled on the 'Virtual Mixer' page. Enable/disable these controls as required.

Media files for the players are saved in two storage locations, either internal storage for messages and chimes, or external storage on a removable microSD card for music and other audio content. The storage location is determined automatically depending on the 'input mode' set for the player on the web app. When the input mode is set to 'MSG' the internal storage will be used, when set to 'BGM' external storage is selected. You can check the current mode selected for each player in the 'Mode' box on the UI. Changing the mode on the web app will update the media content available to the 'Player Source' controls on the demo project.

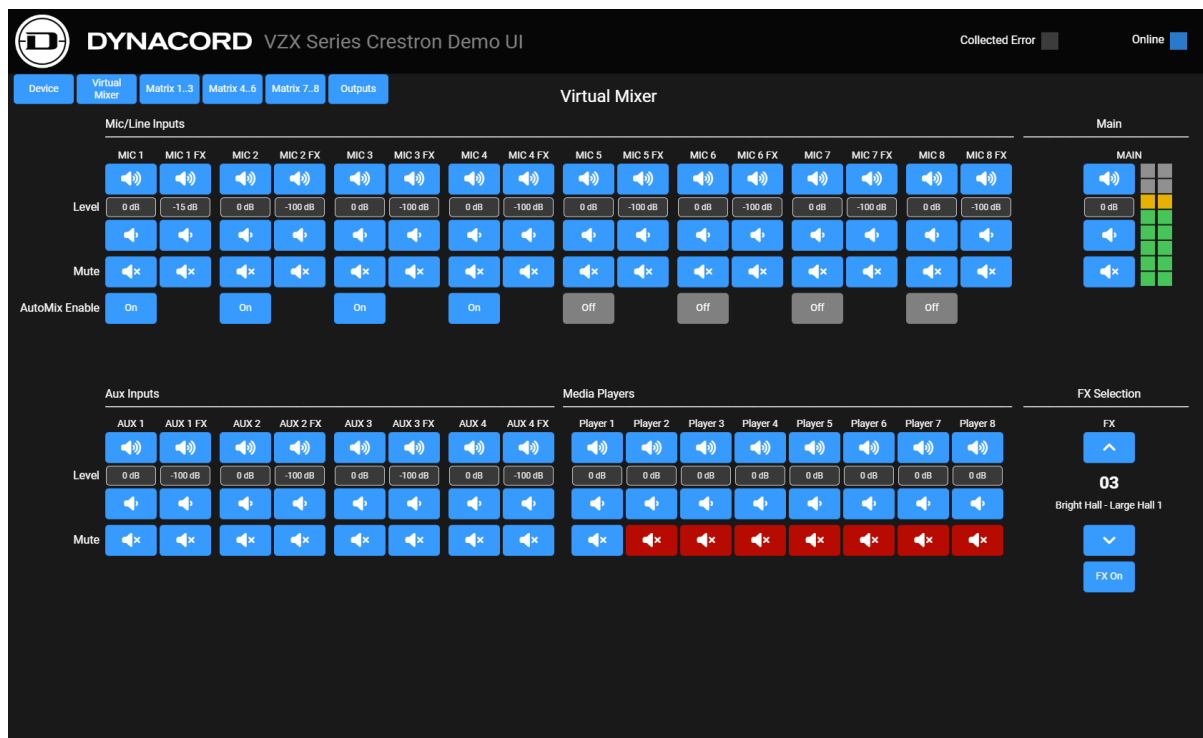
To play media, select the required track or source file using the up/down buttons for the player before clicking the Play button. Once media is started it will continuously loop until stopped. Playing media will also automatically restart whenever the VZX-8 is rebooted. To create playlists of music files see the VZX-8 documentation. Created playlists will be available as a media source when the player is in 'BGM' mode.

The screenshot displays the DYNACORD VZX Series Crestion Demo UI. The interface is divided into several sections:

- Navigation:** Tabs for Device, Virtual Mixer, Matrix 1-3, Matrix 4-6, Matrix 7-8, and Outputs.
- Connection:** Buttons for Connect and Disconnect.
- Device Information:** Fields for Hostname (VZX-18DC50), Sample Rate (48 kHz), FW Version (1.0.1275), Up Time (0d 03h 39m 36s), Current Time (2026/01/28 15:38:28), and Time Zone (Europe/London).
- Network Interface:** Fields for Link Up (checked), Speed (1 Gbit), IP Address (192.168.67.101), Subnet Mask (255.255.255.0), Network Mode (DHCP & Link Local), Default Gateway (192.168.67.1), and MAC Address (00:1C:44:18:DC:50).
- Device Status:** A box indicating the device is "Connected to 192.168.67.101".
- Virtual Mixer Source Selection:** A grid of buttons for FX (L/R), Mix/Line (1-8), Aux (1-4), and Media Player (1-8).
- Media Players:** Eight individual player controls, each with a Mode (BGM), Player Source, Status (PLAYING), and a Play button. The players are currently playing various pop playlists.

Virtual Mixer

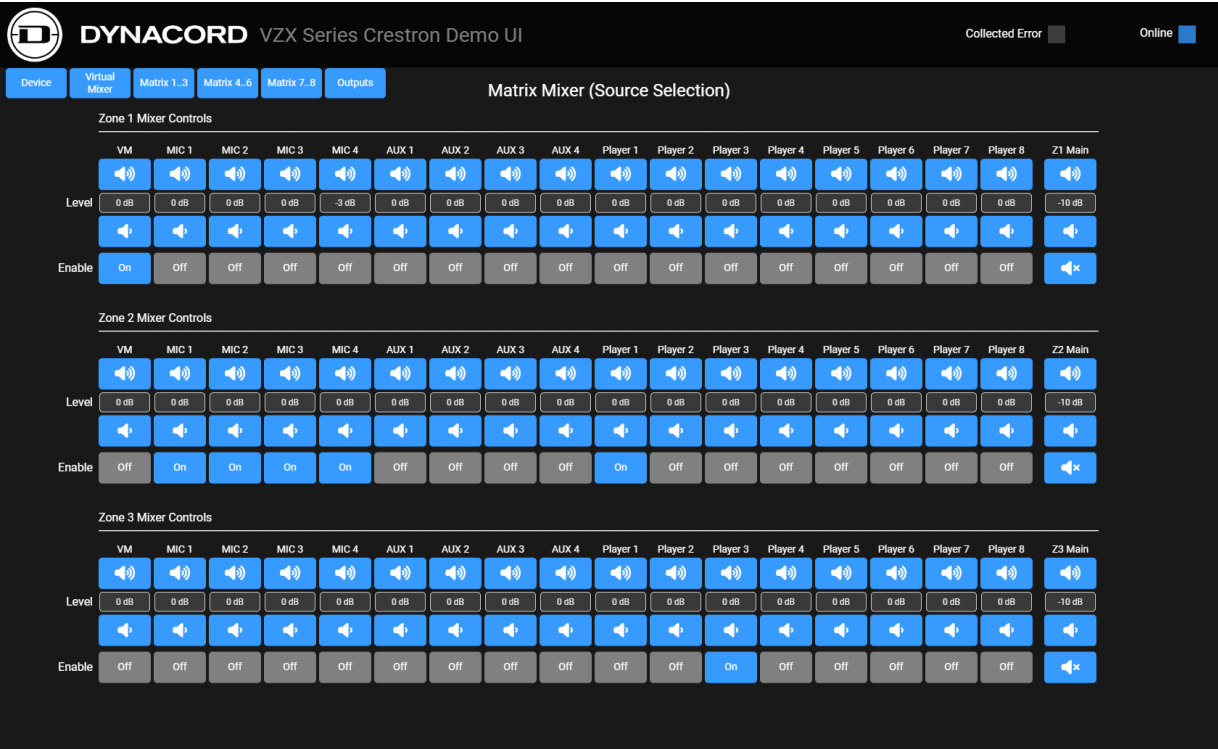
The Virtual Mixer page interacts with some of the more common Virtual Mixer controls you are likely to need in a typical Creston control system. The virtual mixer provides a live mixing interface with digital FX and microphone auto mix capabilities. It can be accessed through properties on the API within your Creston control system, its own web page app using a separate IP address, or the QR code shown on the VZX web app. The virtual mixer can therefore be operated from your Creston control system and/or an iPad simultaneously. The advantage of the virtual mixer is that it allows you to give users access to the mixer without giving access to the full zone configuration. The output of the virtual mixer is a selectable source on the zone matrix mixer allowing you to specify which combination of one or more zones should receive the virtual mixer output mix.



You can specify the virtual mixer input sources on the 'Device' page. Inputs that are not selected will not appear on the VZX-8 virtual mixer app and will be disabled on the Virtual Mixer page on the Creston demo project. The virtual mixer page demonstrates how to control volumes and mutes for each input type, how to control the volume of the FX send level for inputs, select the digital effects program from all of those available on the VZX-8, and how to adjust the main output level. Signal VU metering is also demonstrated for the main output. Although not shown on the demo project, VU metering is also available for each input should you require that for your own projects. Simply follow the code example for the VU output meters on the 'Page2.cs' class file in the Visual Studio solution but using the 'ValueChanged' event for the required meter. For example, the path to the meter for mic input 2 would be `vzx.Device.VirtualMixer.MicInputs[1].VU.ValueChanged` (remember indices for array inputs are zero based in C#).

Zone Matrix Mixer

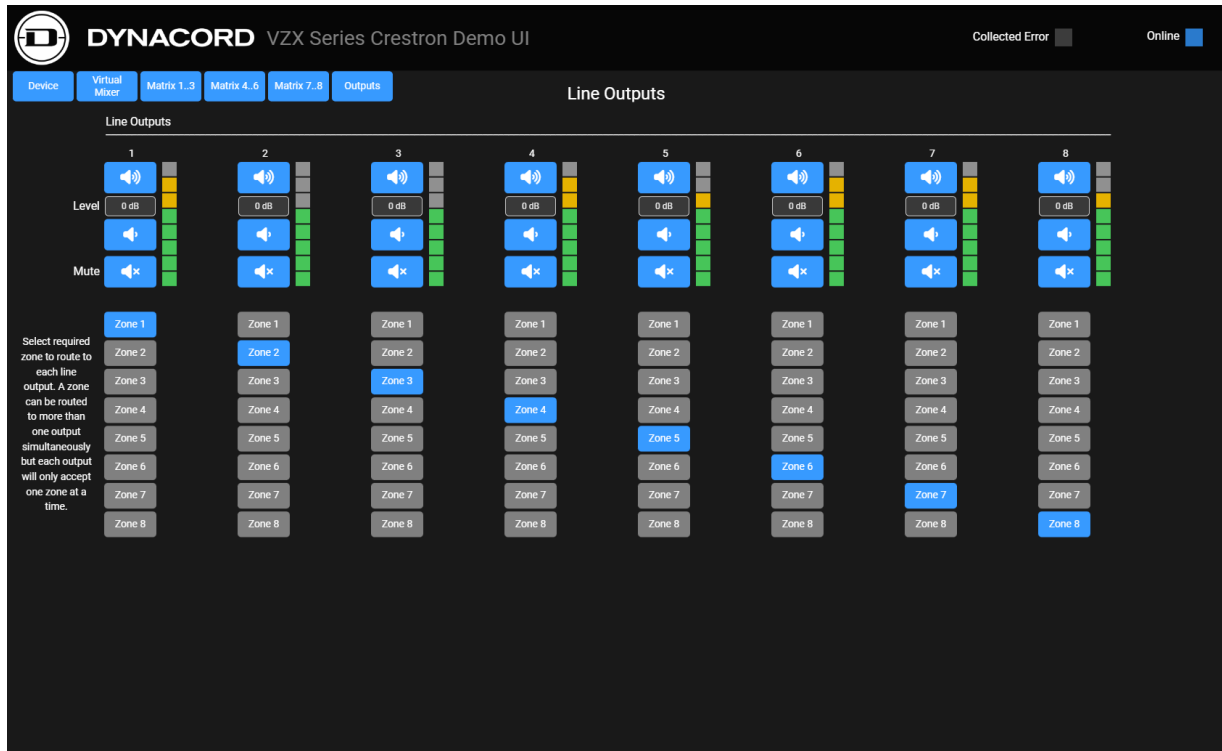
The VZX-8 has eight mixer zones. You can select which input sources can be routed and mixed to each zone independently. Only mic/line inputs 1-4 are shown in the demo project but all eight are available on the API. All of the Aux and media player inputs are demonstrated though, as well as the main output controls for each zone.



The matrix mixer is split over three similar pages where each zone has the same input selection (including volume control and on/off routing to the zone output). Note that you can change the 'input mode' of each input either through the API or by using the web app. Inputs that are set to 'Mix' can be turned on and off individually for a given zone. Inputs that are set to 'BGM' will act as a radio group where only one input can be turned on at a time for the zone. Turning on a different input will automatically turn off the previous one.

Outputs

Each zone can be routed to one or more line outputs simultaneously (although each output will only accept one zone selection at a time). The demo project shows you how to select the zone source for each output, control the line output volume and how to display output VU metering.



3. Using the VZX-8 API in your own projects

This section assumes you will be using the VZX Series API with a Crestron 4-Series or VC-4 control system programmed in C#, with a UI created using Crestron Construct. It outlines the process for creating a control system to remotely manage and control the functions of a VZX-8. 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 programmer skill level required is intermediate to advanced.

The API targets .NET6 as, at the time of development, that was the latest version currently supported by Crestron 4-Series controllers.

You will require:

- A Dynacord VZX-8 audio processor.
- A web browser for initial configuration of the VZX-8 using its web app UI.
- Visual Studio 2022 or 2026 (the free Community edition is also 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 VZX 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 VZX devices.
- Dynacord.VZXSeriesAPI – 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 VZX 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** package version 2.21.184 or later. This package should include **Newtonsoft.Json** package which is also required.
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 .Net 'System' and other namespaces that you may require).

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

2. Let ControlSystem.cs inherit from CrestronControlSystem and add a property for your VZX processor.

```
public class ControlSystem : CrestronControlSystem
{
    public static VZXController? VZX { 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();
        VZX = new VZXController(Model.VZX8, WebSocketScheme.Secure)
        {
            AutoConnect = true,

            //Remember to change the HostName to match your own VZX device!
            HostName = "VZX-18DC50",
            Username = "your user name here",
            Password = "your password here",
        };
        // Start the discovery service to find VZX 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 VZX-8 device

1. The VZX Series API exposes all get/set methods and value changed events for the properties available on the VZX-8 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 a matrix mixer main output, and be notified in your program when the volume changes from another source.

```

public class MyClass
{
    private VZXController vzx = ControlSystem.VZX;

    public MyClass()
    {
        // Example of how to register a value changed event with an inline Lambda
        for (int i = 0; i < Device.NumZones; i++)
        {
            var idx = i // capture i for closure
            vzx.Device.Zones[idx].Mix.MainLevel.ValueChanged += (s, e) =>
            {
                // idx will be the (zero based) index of the matrix mixer output 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
        vzx.Device.Zones[0].Mix.MainLevel.ValueChanged += OnMixerOut1VolumeChanged
    }

    // Example of how to get a property value
    // Property values are only accurate whilst the API is connected to the VZX 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 VZX device, so it does not need to be awaited, as
    // it will return the property value from memory immediately.
    // Note that the idx argument is zero-based.
    public float GetMixerOutputVolume(int idx)
    {
        return vzx.Device.Zones[idx].Mix.MainLevel.GetValue();
    }

    // Example of how to set a property on the remote VZX device
    // ALL SetValue methods are awaited as there will be a small delay while the new value
    // is sent to the VZX 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 SetMixerOutputVolume(int idx, float newVolumeLevel)
    {
        try
        {
            await vzx.Device.Zones[idx].Mix.MainLevel.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 vzx.Device.Zones[idx].Mix.MainLevel.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 the matrix mixer output 1 changes.
    private void OnMixerOut1VolumeChanged (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 (**VZXController**), with a separate instance required for each VZX-8 device you need to control. **VZXController** 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 VZXController named `vzx`

```
private VZXController vxz = new(Model.VZX8, WebSocketScheme.Secure);
```

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

```
public void MyMethod()
{
    string myDeviceFirmware = vxz.Device.FirmwareVersion.GetValue();
    float micInput1Level = vxz.Device.MicInputs[0].Level.GetValue();
    bool virtualMixerPlayer2Mute = vxz.Device.VirtualMixer.PlayerInputs[1].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. The ValueChanged event args `e.Value` will contain the new value of a changed property.

For every settable device property **SetValue(T newValue)** is an async method, so it is highly recommended, and best practice, that your call is always 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. Unhandled exceptions that may occur in an awaited operation will crash your application, which is why it is so important to handle them correctly.

```
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 VZX 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 that all properties implement the same interface to get and set methods and subscribe to value changed events in a consistent way.

Be sure to check out the demo project for practical examples of using numerous different properties exposed by the API.

Class VZXController

This class is the entry point into the API, create an instance of the class for each VZX-8 device you want to control.

Constructor

```
public VZXController(Model model, WebSocketScheme scheme)
```

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 VZXController instance should automatically attempt to connect to the VZX-8 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 VZXController will attempt to automatically connect to the VZX-8 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. Around 3 to 6 seconds is typical, but may be longer if your control system is particularly complex.</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 VZXController instance should automatically attempt to reconnect to the VZX-8 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 VZX 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. 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 connection 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. It takes the form VZX-xxxxxx, where xxxxxx is the last six digits of the device's MAC address.</p>
IsConnected	<p>Indicates whether the VZXController instance is currently connected to a VZX-8 device. When false, and SessionActive is true, the client has lost communications with the device.</p>
IsSessionActive	<p>Indicates whether the VZXController instance session is currently active. When true, the client is connected to, attempting to connect, or attempting to reestablish a connection with, a VZX-8 device.</p>
Model	<p>Gets the model of the VZX-8 this VZXController instance is associated with. As there is currently only one model, this will always return VZX8.</p>
Password	<p>The password configured on the remote device this controller instance is associated with. The Username and Password properties must be set correctly before connecting to the device.</p>
StaticIpAddress	<p>The static IP address of the remote device this VZXController 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 its web app UI. 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>

Username	The username configured on the remote device this controller instance is associated with. The Username and Password properties must be set correctly before connecting to the device.
----------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Methods

ConnectAsync()	<p>Asynchronously connects to the VZX device. Either the StaticIpAddress or HostName property must be set before calling this method. If both properties are set, the StaticIpAddress will take precedence. When using the Hostname, the discovery service must also be running.</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 <code>UserInterface > Page1.cs</code></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 <code>UserInterface > Page1.cs</code></p>
Reboot()	<p>Asynchronously sends a command to reboot the VZX-8 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 the 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> <p>See the demo project <code>ControlSystem.cs</code> class for examples of enabling and using logging.</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</p>
---------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>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 <code>UserInterface > Page1.cs</code></p>
<code>ExternalMediaFilesChanged</code>	<p>Event raised when a change is detected to external media files stored on the microSD card. Raised when files are added, removed, edited, or the SD card is inserted or removed.</p>
<code>InternalMediaFilesChanged</code>	<p>Event raised when a change is detected to internal media files. Raised when files are added, removed or edited.</p>
<code>IsConnectedChanged</code>	<p>Event raised when the connection state changes.</p> <p>If the EventArgs 'Value' property is true, the VZXController instance is successfully connected to the VZX-8 device. If false (and the <code>IsSessionActive</code> property is true), the VZXController instance has lost communications with the device. If false (and the <code>IsSessionActive</code> property is also false), the VZXController 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>
<code>SynchronizationCompleted</code>	<p>Event raised when the initial synchronization of the VZX database is complete after going online.</p> <p>This event can be used by the control system to handle any initialization code that should only be called once the database is fully populated with data from the device. In the demo project, this event is handled to populate media file storage only after it has been retrieved from the device.</p>

VZXController.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

<code>AuxInputs[]</code>	<p>Gets the (zero based) array of auxiliary input channel DSP processing blocks for this device. Auxiliary input one is at index 0, input two at index 1 and so on. The VZX-8 supports four auxiliary inputs.</p>
<code>FirmwareVersion</code>	<p>Gets the firmware version running on the device.</p>
<code>GPIO[]</code>	<p>Gets the (zero based) array of control port general purpose input/output (GPIO) properties for this device. The VZX-8 supports eight GPIO's, each is configurable as an analog input, digital input or digital output.</p>

LineOutputs[]	Gets the (zero based) array of analog line output DSP processing blocks for this device. The VZX-8 supports eight line outputs.
MainFault	Gets the status of the ready/fault relay. This property will be true whenever the device is registering some type of internal fault.
MediaPlayers[]	Gets the (zero based) array of internal media players. Chaining from this property provides access to media player transport controls to start and stop media playback. The VZX-8 supports eight internal media players.
MicInputs[]	Gets the (zero based) array of mic/line input channel DSP processing blocks for this device. The VZX-8 supports eight mic/line inputs.
Name	Gets the host name property returned by the device. This will be the same host name as printed on the information label on the rear panel of the physical VZX-8.
Network	Gets the network configuration properties for this device.
PeriphInputs[]	Gets the (zero based) array of peripheral (accessory) input channel DSP processing blocks for this device. Peripheral inputs are used to control the audio processing of call station microphones. The VZX-8 supports four peripheral inputs.
PlayerInputs[]	Gets the (zero based) array of media player input channel DSP processing blocks for this device. These properties control the audio related functions of a media player. See 'MediaPlayer[]' for properties controlling the audio transport. The VZX-8 supports eight internal media players.
StateFlags	Provides access to various error flags used to monitor individual fault states on the device.
Storage	Provides access to the media storage libraries. Chain from this property to get lists of populated media files for both internal and external (microSD card) storage locations. Each storage location can hold up to 100 media files.
Time	Gets the time properties for this device including up time since the last reboot and time of day.
VirtualMixer	Provides access to virtual mixer controls. Chain from this property to control available inputs on the virtual mixer, the internal digital FX engine, and virtual mixer outputs and VU metering.
Zones[]	Gets the (zero based) array of zones. The 'Mix' property on each zone provides access to the levels and mute controls for each input to the zone. This is one of the most common properties you will likely require when building your control system as you will typically want to control source selection and volume level for the various inputs within a given zone. The VZX-8 supports eight zones.

Methods

Clone commands	<p>Multiple methods to asynchronously copy configuration settings from one DSP block to another block of the same type. For example, call the method <code>CloneAuxInputEqConfig(int originInput, int destinationInput)</code> to copy EQ settings from one Aux input to another.</p> <p>Although these methods are available they would not commonly be used with a control system.</p> <p>The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.</p>
----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

VZXController.Device.MicInputs[]

Access to the array of input based properties. You must specify the (zero based) index of the input you want to access → `VZXController.Device.MicInputs[0]` etc. The properties for mic/line inputs are described here but Aux, Peripheral and Media Player inputs all have the same properties, methods and events except for **GainInDb** and **PhantomPower** which are only available on mic/line inputs.

Properties

AGC	Gets the AGC DSP block for this microphone input. Chain from AGC to access its properties.
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 (zero based) array of EQ DSP blocks for this microphone input. There are four EQ filters available. Chain from <code>EQ[idx]</code> to access its properties.
GainInDb	Gets the mic gain property (in dB) for this microphone input. The allowed range is 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 -100.0 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.
VU	Gets the VU meter level for this microphone input. Handle the ValueChanged event to be informed whenever a VU meter value changes.

Methods

SetDspConfig(DsplInputProcessingConfig)	<p>Asynchronously 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>
SetMode(MicInputMode)	<p>Asynchronously sets the operational mode for the input. Note: different input types (Mic, Aux, Player etc.) may have different options from the list below.</p> <ul style="list-style-type: none"> • BGM – only one BGM source can be enabled at a time on the zone matrix mixer for any given zone. Enabling a different BGM source will automatically disable the previous selection. Use this mode for music where it makes sense that only one source should ever be playing out into the zone. • MIX – multiple inputs can be enabled on the zone mixer for a given zone. Useful for microphones and other sources where more than one can be mixed together to play out into the zone. • VOX – when audio is detected on this input the ducker for the input can be used to reduce or mute the level of all other inputs whilst the audio is present. Useful for paging announcements to override background music playing into the zone. • ANC – (only for mic/line inputs) the input can be used to detect background noise in an area. The input can then be selected as the ambient input of the ANC (ambient noise compensation) DSP block on a line output to automatically control the output level depending on the amount of noise detected. • MSG – (only for media player inputs) enables the media player to be used to play chimes and messages from the internal media storage. When a media player mode is BGM it can play music files and playlists from the external (microSD) storage.

VZXController.Device.MediaPlayers[]

Access to the array of media player properties. You must specify the (zero based) index of the player you want to access → VZXController.Device.MediaPlayers[0] etc. There are eight internal media players available.

Properties

ActiveMedia	ActiveMedia is the full file path, including the name, of the file currently being played by the media player. See the demo project 'MediaPlayerMap.cs' file for an example of simple string manipulation to remove the file path information from the string value to display the active media file name.
Enabled	Gets the enabled property for the media player instance.
Status	Gets current status of the media player instance. Idle, Playing, Stopped etc.

Methods

StartMediaPlayback(file)	Asynchronously starts playback of a media file from the external storage (microSD). The 'file' argument must be the full path to the required media file on the microSD card. See the demo project 'MediaPlayerMap.cs' file for an example of obtaining the file path and starting media playback. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
StopMediaPlayback()	Asynchronously stops playback of a media file from the external storage (microSD). See the demo project 'MediaPlayerMap.cs' file for an example of stopping media playback. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
StartMessagePlayback(file)	Asynchronously starts playback of a message file from the internal storage. The 'file' argument must be the full path to the required media file in the internal storage. See the demo project 'MediaPlayerMap.cs' file for an example of obtaining the file path and starting message playback. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.
StopMessagePlayback()	Asynchronously stops playback of a media file from the internal storage. See the demo project 'MediaPlayerMap.cs' file for an example of stopping message playback. The caller of this method is responsible for handling any exceptions that may be thrown during the awaited operation.

VZXController.Device.Storage

Access to the storage subsystem on a VZX-8. Storage is where media files are located, either internal storage for chimes and messages or external storage on a microSD card for music and other audio content (including playlists). See the demo project 'ControlSystem.cs' file for an example of using the storage properties and methods.

Properties

ExternalDetected	Indicates whether an external microSD card is present on the VZX-8.
ExternalMediaStorageFiles[]	Gets the (zero based) array of media files in external storage. Up to 100 media files can be loaded onto the microSD card. Check the 'ExternalMediaStorageFiles[idx].Populated' property to determine whether an element in the array is actually populated with a media file.
InternalMediaStorageFiles[]	Gets the (zero based) array of media files in internal storage. Up to 100 message, chime and alarm tone files can be loaded onto the internal storage. Check the 'InternalMediaStorageFiles[idx].Populated' property to determine whether an element in the array is actually populated with a media file.

Methods

GetPopulatedExternalMediaFiles()	Returns a list of all media files available on the external storage. This method inspects the ExternalMediaStorageFiles[] array, removing any elements within the array that are not populated. Each item in the list is a MediaPlayerFile instance containing properties for the name, path, duration and file format of the media file.
GetPopulatedInternalMediaFiles()	Returns a list of all media files available on the internal storage. This method inspects the InternalMediaStorageFiles[] array, removing any elements within the array that are not populated. Each item in the list is a MediaPlayerFile instance containing properties for the name, path, duration and file format of the media file.

VZXController.Device.VirtualMixer

Access to the virtual mixer. See the demo project 'Page2.cs' file for examples of how to control the virtual mixer.

Properties

AuxInputs[]	Gets the (zero based) array of aux inputs to the virtual mixer. Use the AuxInputs[idx].Level property to adjust the level of this source on the virtual mixer. The allowed range is between -100.0 dB and 10.0 dB Use the AuxInputs[idx].Selected property to enable/disable the input on the virtual mixer.
FX	Gets the FX library properties. See the demo project 'Page2.cs file' for examples of selecting and loading digital effects.
Level	Gets the main output level property (in dB) for the virtual mixer. The allowed range is between -100.0 dB and 10.0 dB

MicInputs[]	<p>Gets the (zero based) array of mic/line inputs to the virtual mixer.</p> <p>Use the MicInputs[idx].Level property to adjust the level of this source on the virtual mixer. The allowed range is between -100.0 dB and 10.0 dB</p> <p>Use the MicInputs[idx].Selected property to enable/disable the input on the virtual mixer.</p>
Mute	Gets the main output mute property for the virtual mixer.
Name	Gets the name property for the virtual mixer.
PlayerInputs[]	<p>Gets the (zero based) array of media player inputs to the virtual mixer.</p> <p>Use the PlayerInputs[idx].Level property to adjust the level of this source on the virtual mixer. The allowed range is between -100.0 dB and 10.0 dB</p> <p>Use the PlayerInputs[idx].Selected property to enable/disable the input on the virtual mixer.</p>
VU_Left	Gets the left output VU meter level for the virtual mixer. Handle the ValueChanged event to be informed whenever a VU meter value changes.
VU_Right	Gets the right output VU meter level for the virtual mixer. Handle the ValueChanged event to be informed whenever a VU meter value changes.

VZXController.Device.Zones[]

Access to the zone matrix mixer. You must specify the (zero based) index of the zone you want to access → VZXController.Device.Zones[0] etc. The VZX-8 has eight zones.

Properties

Ducker	Gets the Ducker DSP block for this zone. Chain from Ducker to access its properties.
Mix	See below.
Name	Gets the name property for this zone.

VZXController.Device.Zones[idx].Mix

Access to the zone matrix mixer. See the demo project 'Page3.cs' and 'CrosspointWidgetMap.cs' files for examples of how to control the zone matrix mixer.

Properties

AuxInputs[]	<p>Gets the (zero based) array of aux inputs to the mixer for this zone.</p> <p>Use the AuxInputs[idx].Level property to adjust the level of this source on the referenced the zone. The allowed range is between -100.0 dB and 10.0 dB</p> <p>Use the AuxInputs[idx].Mute property to turn this source on or off on the referenced zone.</p>
MainLevel	<p>Gets the main output level property (in dB) for this zone. The allowed range is between -100.0 dB and 10.0 dB</p>
MainMute	<p>Gets the main output mute property for this zone.</p>
MicInputs[]	<p>Gets the (zero based) array of mic inputs to the mixer for this zone.</p> <p>Use the MicInputs[idx].Level property to adjust the level of this source on the referenced the zone. The allowed range is between -100.0 dB and 10.0 dB</p> <p>Use the MicInputs[idx].Mute property to turn this source on or off on the referenced zone.</p>
PlayerInputs[]	<p>Gets the (zero based) array of media player inputs to the mixer for this zone.</p> <p>Use the PlayerInputs[idx].Level property to adjust the level of this source on the referenced the zone. The allowed range is between -100.0 dB and 10.0 dB</p> <p>Use the PlayerInputs [idx].Mute property to turn this source on or off on the referenced zone.</p>
VirtualMixer	<p>Gets the virtual mixer properties related to this zone.</p> <p>Use the VirtualMixer.Level property to adjust the level of this source on the referenced the zone. The allowed range is between -100.0 dB and 10.0 dB</p> <p>Use the VirtualMixer.Mute property to turn this source on or off on the referenced zone.</p>

VZXController.Device.LineOutputs[]

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

Properties

ANC	Gets the ANC (ambient noise compensation) DSP block for this line output. Chain from ANC to access its properties.
Delay	Gets the Delay DSP block for this line output. Chain from Delay to access its properties.
EQ[]	Gets the (zero based) array of EQ DSP blocks for this line output. There are five EQ filters available on each output. Chain from EQ[idx] to access its properties.
Level	Gets the level property (in dB) for this line output. The allowed range is between -100.0 dB and 10.0 dB
Linked	Gets the linked property for this line output. Only valid for (zero based) even line outputs (0, 2, 4, 6), will be null for odd outputs (1, 3, 5, 7).
Mute	Gets the mute property for this line output.
Name	Gets the name property for this line output.
VU	Gets the VU meter level for this line output. Handle the ValueChanged event to be informed whenever a VU meter value changes.

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. 2026