****

**Crestron Certified Drivers**

Framework Overview

Crestron Electronics, Inc.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Comments | Author |
| 1.00.00 | 10/2/2018 | First draft | Casey Blicharz |
| 1.00.01 | 1/25/2018 | Added more sections and correcting grammar mistakes | Casey Blicharz |
| 1.00.02 | 2/14/2018 | Split document and added more information | Casey Blicharz |
| 1.00.03 | 7/17/2019 | Added security system information to the document | Natalie Jackman |
| 1.00.04 | 9/13/2019 | Adding MediaServiceProvider, ArtworkMode, and DisplayMode | Casey Blicharz |

Crestron and the Crestron logo are either trademarks or registered trademarks of Crestron Electronics, Inc. in the United States and/or other countries. Other trademarks, registered trademarks, and trade names may be used in this document to refer to either the entities claiming the marks and names or their products. Crestron disclaims any proprietary interest in the marks and names of others. Crestron is not responsible for errors in typography or photography.

Table of Contents

[1 Framework 4](#_Toc23857186)

[1.1 Libraries 5](#_Toc23857187)

[1.2 Driver Class Definition 6](#_Toc23857188)

[1.3 Device Type Interfaces 8](#_Toc23857189)

[1.3.1 Video Displays and Projectors 9](#_Toc23857190)

[1.3.2 Cable Boxes 10](#_Toc23857191)

[1.3.3 Video Servers 11](#_Toc23857192)

[1.3.4 Blu-ray Players 12](#_Toc23857193)

[1.3.5 AV Receivers 14](#_Toc23857194)

[1.3.6 Security Systems 14](#_Toc23857195)

[1.4 Device Type Abstract classes 16](#_Toc23857196)

[1.4.1 ABasic<Device Type> 16](#_Toc23857197)

[1.4.2 ATransportDriver And Transports 17](#_Toc23857198)

[1.4.3 A<Device Type>Protocol 21](#_Toc23857199)

[1.4.4 Driver JSON Data 50](#_Toc23857200)

# Framework

The framework consists of several libraries that provide a set of interfaces that define how drivers function and a set of classes that provide common code for all drivers to use.   
  
It provides a set of standard features and types of feedback for several device types, and default implementations of all features supported by a driver. It also provides some default transport classes for applications and drivers to use for connecting to the device.

Device types are defined by interfaces that are defined by several component interfaces. Each device type has an abstract class associated with it that provides the default implementations of all its component interfaces.

The following sections will go over the several libraries that make up the framework, the basic makeup of a driver, device type interfaces that define the functionality of a driver, the abstract classes drivers use to provide default functionality and the driver itself which overrides anything where the default functionality needs a modification.

## Libraries

Writing and loading drivers in SIMPL Windows and in S# Pro applications require the use of the following libraries:

|  |  |
| --- | --- |
| Name | Description |
| RADCommon | Provides a definition of all device types, standard commands and features, commonly used transports, and logic for features supported by all drivers. |
| RADDisplay | Provides the framework for use by flat panel display and projector drivers. |
| RADCableBox | Provides the framework for use by cable box and satellite drivers. |
| RADAvReceiver | Provides the framework for use by AV receiver drivers. |
| RADBlurayPlayer | Provides the framework for use by Blu-ray player drivers. |
| RADVideoServer | Provides the framework for use by video server drivers. |
| RADSecuritySystem | Provides the framework for use by security system drivers. |

|  |  |
| --- | --- |
| S# Pro Only | |
| **Name** | **Description** |
| RADProTransports | Provides transport classes for use in an S# Pro environment. |
| IrDisplay | An IR driver that implements ABasicVideoDisplay |
| IrCableBox | An IR driver that implements ABasicCableBox |
| IrAvReceiver | An IR driver that implements ABasicAVReceiver |
| IrBlurayPlayer | An IR driver that implements ABasicBlurayPlayer |
| IrVideoServer | An IR driver that implements ABasicVideoServer |
| CrestronConnected Display/Projector | Crestron Connected drivers that implement ABasicVideoDisplay |

These libraries provide all the interfaces, abstract classes, and classes required for creating a driver for use on Crestron systems.

## Driver Class Definition

Abstract Device Type Class

Abstract Driver Class

Abstract Driver  
Protocol Class

Abstract  
 Transport Class

Abstract Device Type Protocol Class

Driver Protocol Class

Main  
Driver Class

Driver Transport Class

Device Type Response Validation Class

Driver Response Validation Class

Driver JSON Data

**The required structure of a driver**

Inheritance

Definition

The main driver class is responsible for providing the application methods that will control the device, providing feedback when the device provides it, and providing events that let applications know when feedback changes. Various classes handle this control and feedback internally.  
  
 It will implement all of the required component interfaces its device type interface defines and the framework handles most of these implementations. It will define an Initialize method from one of several transport interfaces explained later to allow applications to set up the driver’s communication properties.

The transport class is responsible for sending data to the device and routing received messages to the protocol class. Drivers can create their own transport if needed or they can use one of the available ones in the common libraries.

The protocol class is responsible for building and sending commands to the transport class, properly pacing the commands prior to sending, and for setting up response validation.

The response validation class is responsible for parsing data received in order to validate a response. It will check for delimiters, if available, to find the full message and then parse it to find out what kind of message it is, such as power feedback, and will provide the API code for the new state.

The JSON driver data contains most of the API for the device, meta-data about the device, communication settings on the device, and features that are supported. All the classes listed above consume this data.

Applications will only need knowledge of the device type interfaces to make calls to the driver and to handle state change events.

## Device Type Interfaces

The current supported device types are:

* Video displays and projectors
* Cable boxes
* Video servers
* Blu-ray Players
* AV Receivers
* Security Systems

Each device type inherits from several component interfaces that define specific features. Appendix A defines these interfaces.  
  
As new component interfaces are added to support new features, new device type interfaces are added and versioned by a number in their name. The abstract device type class will not change names and will be updated to implement the newest device type interface.

### Video Displays and Projectors

|  |
| --- |
| public interface IBasicVideoDisplay6 : IBasicVideoDisplay5, IMediaServiceProvider, IKeypad, IDvr, IDvr2, IArtwork, IVideoConfiguration, IChannel  {  bool SupportsLetterKeys { get; }  List<LetterButtons> LetterButtonsSupported { get; }  void LetterKeys(LetterButtons letterKeys);  bool SupportsColorButtons { get; }  List<ColorButtons> ColorButtonsSupported { get; }  void ColorButton(ColorButtons color);  } |

|  |
| --- |
| public interface IBasicVideoDisplay5 : IBasicVideoDisplay4, IModelFileSupport  { } |

|  |
| --- |
| public interface IBasicVideoDisplay4 : IBasicVideoDisplay3, ISupportedCommandsHelper  { } |

|  |
| --- |
| public interface IBasicVideoDisplay3 : IBasicVideoDisplay2, IMediaSource, IMediaSource2, IMediaTransport, IMediaTransport2, IMediaTransportInfo, IMediaTransportOptions  { } |

|  |
| --- |
| public interface IBasicVideoDisplay2  : IBasicVideoDisplay, IInputs2, IOutputs, IConnection2, INavigation, INavigation2, IBasicInformation2  { } |

|  |
| --- |
| public interface IBasicVideoDisplay  : IBasicInformation, IPower, IInputs,  IVolume, ICustomCommand, ICustomCommandCollection,  IConnection, IEnergyStar, IVideoMute, ILampHours, IBasicLogger, IDisposable  {  event Action<DisplayStateObjects, IBasicVideoDisplay, byte> StateChangeEvent;  } |

### Cable Boxes

|  |
| --- |
| public interface IBasicCableBox5 : IBasicCableBox4, IModelFileSupport  { } |

|  |
| --- |
| public interface IBasicCableBox4 : IBasicCableBox3, ISupportedCommandsHelper  { } |

|  |
| --- |
| public interface IBasicCableBox3 : IBasicCableBox2, IDvr2  {  bool SupportsOnDemand { get; }  void OnDemand();  } |

|  |
| --- |
| public interface IBasicCableBox2 : IBasicCableBox, IOutputs, IConnection2, IMediaTransport2, IBasicInformation2, IMediaTransportOptions  { } |

|  |
| --- |
| public interface IBasicCableBox  : IBasicInformation, IPower, IVolume, INavigation,  IKeypad, IDvr, IMediaTransport, IChannel,  ICustomCommand, ICustomCommandCollection,  IBasicLogger, IConnection, IEnergyStar, IDisposable  {  bool SupportsLetterKeys { get; }  List<LetterButtons> LetterButtonsSupported { get; }  void LetterKeys(LetterButtons letterKeys);  bool SupportsColorButtons { get; }  List<ColorButtons> ColorButtonsSupported { get; }  void ColorButton(ColorButtons color);  bool SupportsFavorite { get; }  void Favorite();  bool SupportsInfo { get; }  void Info();  bool SupportsLast { get; }  void Last();  bool SupportsReplay { get; }  void Replay();  bool SupportsThumbsUp { get; }  void ThumbsUp();  bool SupportsThumbsDown { get; }  void ThumbsDown();  event Action<CableBoxStateObjects, IBasicCableBox, byte> StateChangeEvent;  } |

### Video Servers

|  |
| --- |
| public interface IBasicVideoServer6 : IBasicVideoServer5, IModelFileSupport  { } |

|  |
| --- |
| public interface IBasicVideoServer5 : IBasicVideoServer4, ISupportedCommandsHelper  { } |

|  |
| --- |
| public interface IBasicVideoServer4 : IBasicVideoServer3, INavigation3  { } |

|  |
| --- |
| public interface IBasicVideoServer3 : IBasicVideoServer2, IMediaSource, IMediaSource2  {  bool SupportsReplay { get; }  void Replay();  } |

|  |
| --- |
| public interface IBasicVideoServer2  : IBasicVideoServer, INavigation2, IOutputs, IConnection2, IPower, IMediaTransport2, IBasicInformation2, IMediaTransportOptions, IMediaTransportInfo  { } |

|  |
| --- |
| public interface IBasicVideoServer  : IBasicInformation, INavigation, IKeypad, IMediaTransport,  ICustomCommand, ICustomCommandCollection, IConnection, IBasicLogger, IDisposable  {  event Action<VideoServerStateObjects, IBasicVideoServer, byte> StateChangeEvent;  } |

### Blu-ray Players

|  |
| --- |
| public interface IBasicBlurayPlayer4 : IBasicBlurayPlayer3, IModelFileSupport  { } |

|  |
| --- |
| public interface IBasicBlurayPlayer3 : IBasicBlurayPlayer2, ISupportedCommandsHelper  { } |

|  |
| --- |
| public interface IBasicBlurayPlayer2 : IBasicBlurayPlayer, IOutputs, IConnection2, IMediaTransport2, IBasicInformation2, IMediaTransportInfo  {  bool SupportsReplay { get; }  void Replay();  bool SupportsColorButtons { get; }  } |

Continued on next page…

|  |
| --- |
| public interface IBasicBlurayPlayer  : IBasicInformation, IPower, IMediaTransport,  INavigation, IKeypad, ICustomCommand, IConnection,  ICustomCommandCollection, IEnergyStar, IBasicLogger, IDisposable  {  event Action<BlurayPlayerStateObjects, IBasicBlurayPlayer, byte> StateChangeEvent;  bool SupportsPlayBackStatusFeedback { get; }  PlayBackStatus PlayBackStatus { get; }  bool SupportsTrackFeedback { get; }  string TrackFeedback { get; }  bool SupportsChapterFeedback { get; }  string ChapterFeedback { get; }  bool SupportsTrackElapsedTimeFeedback { get; }  TimeSpan TrackElapsedTime { get; }  bool SupportsTrackRemainingTimeFeedback { get; }  TimeSpan TrackRemainingTime { get; }  bool SupportsChapterElapsedTimeFeedback { get; }  TimeSpan ChapterElapsedTime { get; }  bool SupportsChapterRemainingTimeFeedback { get; }  TimeSpan ChapterRemainingTime { get; }  bool SupportsTotalElapsedTimeFeedback { get; }  TimeSpan TotalElapsedTime { get; }  bool SupportsTotalRemainingTimeFeedback { get; }  TimeSpan TotalRemainingTime { get; }  bool SupportsAudio { get; }  void Audio();  List<ColorButtons> ColorButtonsSupported { get; }  void ColorButton(ColorButtons color);  bool SupportsDisplay { get; }  void Display();  bool SupportsEject { get; }  void Eject();  void PopUpMenu();  bool SupportsPopUpMenu { get; }  void TopMenu();  bool SupportsTopMenu { get; }  bool SupportsOptions { get; }  void Options();  bool SupportsSubtitle { get; }  void Subtitle();  } |

### AV Receivers

|  |
| --- |
| public interface IBasicAvr4 : IBasicAvr3, IModelFileSupport  { } |

|  |
| --- |
| public interface IBasicAvr3: IBasicAvr2, ISupportedCommandsHelper  { } |

|  |
| --- |
| public interface IBasicAvr2 : IBasicAvr, INavigation2, IMediaSource2  { } |

|  |
| --- |
| public interface IBasicAvr  : IBasicInformation, IConnection, IPower, IOutputs,  IInputs2, IAudio, IVolume, ISurround, ITuner,  INavigation, IMediaSource, IBasicLogger, IDisposable,  IBasicInformation2  {  List<IAvrZone> AvrZones { get; set; }  event Action<AvrStateObjects, IBasicAvr, byte> StateChangedEvent;  } |

### Security Systems

|  |
| --- |
| public interface IBasicSecuritySystem2 : IBasicSecuritySystem, ISecurityResourceControl, ISecurityDeviceLogging, IDeviceVersioning  {  event Action<SecuritySystemStateObjects, object> ObjectStateChangeEvent;  bool SupportsLogin { get; }    bool SupportsRequestAreasNotReadyToArm { get; }  void RequestAreasNotReadyToArm(List<Resource> Resources);    void Login(string Username, string password);  void Login(string password);  } |

Continued on next page…

|  |
| --- |
| public delegate void SecuritySystemEventsEventHandler(SecuritySystemStatus securitySystemStatus);  public interface IBasicSecuritySystem  : IBasicInformation, IKeypad, INavigation, IAuxiliaryButtons,  ICustomCommand, ICustomCommandCollection,  IBasicLogger, IConnection, IDisposable  {  bool SupportsKeypadStatusText { get; }  string KeypadStatusText { get; }  Dictionary<int, SecuritySystemKeypadLed> KeypadLeds { get; }  Dictionary<SecuritySystemStatus, bool> CurrentSystemStates { get; }  SecuritySystemSettableStates[] SettableSystemStates { get; }    void SetSystemState(SecuritySystemSettableStates stateToSet);    bool SupportsUdlPassword { get; }  string UdlPassword { get; set; }    event Action<SecuritySystemStateObjects, byte> StateChangeEvent;  }  public class SecuritySystemKeypadLed  {  public bool State { get; set; }  public SecuritySystemKeypadLedColors Color { get; set; }  public string Label { get; set; }  } |

## Device Type Abstract classes

### ABasic<Device Type>

Every device type interface will have an abstract class that implements it and all the necessary component interfaces and provides basic functionality. Every abstract device type class internally inherits the same abstract common driver class called ABasicDriver that handles the common logic shared by all device types.

This class will contain a property for the protocol and transport classes. The protocol class is responsible for the API and the transport class is responsible for communication with the device.

The only two component interfaces that must be overridden here are IAuthentication and IAuthentication2. Later sections will detail how this is done.

The methods defined by this class handle:

* Driver JSON data deserialization
* Populating lists of supported commands
* Routing of all API calls to the protocol class
* Handling all state changes reported by the protocol class
* Handling internal meta-data

Every driver written is required to implement the correct device type abstract class.

|  |  |
| --- | --- |
| Device Type | Abstract Class |
| Display / Projector | ABasicVideoDisplay |
| Cable Box | ABasicCableBox |
| Blu-ray Player | ABasicBlurayPlayer |
| Video Server | ABasicVideoServer |
| AV Receiver | ABasicAvReceiver |
| Security System | ABasicSecuritySystem |

### ATransportDriver And Transports

Every driver will only support one transport and will define what transport it supports using one of the several transport interfaces. The driver class is required to implement one of these interfaces. If the transport chosen requires hardware access then ISimpl must also be defined to support the driver in SIMPL Windows if possible.

All transports are of type ATransportDriver. While ATransportDriver does provide some logic that is common to all transports such as logging, the actual operation of the driver’s transport class needs to do a few things:

|  |  |
| --- | --- |
| Required custom implementation | |
| DataHandler | Must be called when the transport receives data. The protocol class will handle assignment to this and should not be done by drivers. |
| ConnectionChanged | Must be called when the transport detects a disconnect if possible. |
| IsConnected | Must be set to true when communication is possible with the device and false when it is not. |
| SendMethod | Must be overridden by the driver’s transport class. This method must send data to the device with no added delay. |
| Start | Must be overridden to connect to the device. |
| Stop | Must be overridden to disconnect from the device.  Since Stop() will be called by the protocol when it is disposed, this method must dispose any IDisposable classes. |
| Log | Must be called when a message needs to be logged such as a change of states. |

The transport should also behave in an asynchronous way In addition to implementing the methods above.

All other members of ATransportDriver may not be modified, called, or used.

There are several pre-built transports available for driver developers to use. The next section goes over the interfaces that define the transport of a driver and what pre-built transport exist to support that type.

#### ITcp

|  |
| --- |
| public interface ITcp : ISerial  {  int Port { get; }  void Initialize(IPAddress ipAddress, int port);  } |

Used by all TCP/IP drivers which require an IP address and a port. Applications are responsible for resolving hostnames to an IP address.

The port value is zero by default and is set by the defined Initialize method. Applications may call (driverObject as ITcp).Port to get the default value defined in the driver’s JSON data.

TcpTransport and HttpTransport are two pre-built transports that may be used but are not required. TcpTransport is for drivers that have a constant connection to the device. It handles reconnects if the driver’s JSON data set EnableAutoReconnect to true every minute. This transport is not meant to be used by drivers that need to open a connection, send a command, and then close the connection. A custom transport would need to be programmed for these cases.

HTTPS, SSL\*, SSH, WSS, and other protocols must be implemented by the driver with its own custom transport class.

\*The security system device type can utilize the variant of the TCP transport called TcpSSLTransport.

#### ISimpl

|  |
| --- |
| public interface ISimpl : ISerial  {  SimplTransport Initialize(Action<string, object[]> send);  } |

Required when the transport requires hardware access such as COM drivers. The driver will call the provided Send method when data needs to be sent to the device.

SimplTransport must be used as the transport type when ISimpl needs to be implemented. It contains a method called “ReceiveData” that needs to be called when data is received. The SIMPL modules use this class for IR, CEC, and COM drivers.

Some S# Pro applications may choose to use this transport so that they could handle the COM port on their own so it is important this is implemented if the driver requires hardware access.

#### ISerialComport

|  |
| --- |
| public interface ISerialComport : ISerial  {  ComPortSpec ComSpec { get; }  void Initialize(IComPort comPort);  } |

Required by all COM drivers in S# Pro applications. The driver must also inherit and implement ISimpl.

The ComSpec is defined by the driver’s JSON data and is set automatically. The COM port object passed in by Initialize should have its COM specifications set using the settings found in ComSpec if the application wants to use the default configuration.

The class SerialTransport found in RADProTransports.dll must be used for this transport type by S# Pro applications.

#### ICecDevice

|  |
| --- |
| public interface ICecDevice  {  void Initialize(ISerialTransport transport);  SimplTransport Initialize(int id, Action<string, object[]> send);  } |

Required by all CEC drivers. Drivers have to implement both Initialize methods to support SIMPL Windows and S# Pro.

The class CecTransport found in RADProTransports.dll must be used for this transport type by S# Pro applications.

#### ICrestronConnected

|  |
| --- |
| public interface ICrestronConnected  {  void Initialize(uint ipid, CrestronControlSystem controlSystem);  } |

Required by all Crestron Connected drivers. This driver can only be used in S# Pro so there is no need for defining ISimpl as a secondary transport type.

#### IIr

|  |
| --- |
| public interface IIr  {  void Initialize(IIrPort port, string pathToIrFile);  void Initialize(IIrPort port);  void Initialize(string pathToIrFile);  } |

Required by all IR drivers used in S# Pro.

These initialize methods support both a PKG file and an IR file. If a PKG file is given but not an IR port then the driver will “preview” what it supports by using the DAT file contained in the PKG file. If the port is given then the IR file will determine what the driver supports.

The framework provides the following pre-built IR drivers so that developers only need to create the IR file or the PKG file that contains the IR file.

|  |  |
| --- | --- |
| Driver Library | Function |
| IrDisplay | Display and projectors |
| IrCableBox | Cable boxes |
| IrAvReceiver | AV receivers |
| IrBlurayPlayer | Blu-ray players |
| IrVideoServer | Video servers |

The security system device type does not support IR drivers.

### A<Device Type>Protocol

Every abstract device type class will include a reference to an abstract protocol class. These abstract classes inherit from ABaseDeviceProtocol that handles the common logic shared between all the device type protocols (explained later). The driver’s protocol class must implement this class.

This class is responsible for the following:

* Supplying needed methods for handling specific API constraints.
* Providing virtual methods that implement the device type interface which build and send the appropriate commands. The structure CommandSet is used for these.
* Managing custom commands, user attributes, and authentication
* Handling validated data from the Response Validation class. The structure ValidatedRxData is used for this.
* Handling the driver-defined polling sequence.
* Command pacing when sending data to the transport class.

By default, all commands are built based on the driver JSON data. The majority of virtual methods in the protocol class have the same names as the component interfaces’ methods. Drivers may override these methods to build commands manually. Section 2 will explain how to do this.

Every driver written is required to implement the correct protocol class:

|  |  |
| --- | --- |
| Device Type | Abstract Class |
| Display / Projector | ADisplayProtocol |
| Cable Box | ACableBoxProtocol |
| Blu-ray Player | ABlurayPlayerProtocol |
| Video Server | AVideoServerProtocol |
| AV Receiver | AReceiverProtocol |
| Security System | ASecuritySystemProtocol |

#### Override Methods

Apart from the component interface methods, there also other methods that can be overridden if the default functionality is not desired due to API constraints. Any methods not listed may not be overridden.

|  |  |
| --- | --- |
| Method | Description & Use |
| Dispose | Implementation of IDisposable. Call the base after complete. |
| ConnectionChanged | Invoked when the connection state changes. Call the base after complete. |
| MessageTimedOut | Invoked when a message is considered to be timed out. Only polling messages can time out. |
| ChooseDeconstructMethod | Invoked when the validated data from Response Validation is marked as ready. Base may only be called if the driver did not do anything with the validated data.  The default implementation of this method is a switch statement on the command group of the ValidatedRxData (Section 1.4.2.5) and calls the appropriate Deconstruct<Feature> method in the protocol code. |
| Deconstruct<Feature> | Invoked by the base implementation of ChooseDeconstructMethod.  The default implementations of these methods will attempt to compare the data within the ValidatedRxData object to feedback data within the driver JSON data to match it to a particular response. It will then update feedback internally, call Log with the new state, call FireEvent with the correct event type.  An example of this would be a PowerPoll response where Data would be “\x01” and the command group would be “Power”. DeconstructPower would set PowerIsOn to true in the protocol and call FireEvent(DisplayStateObjects.Power).  Drivers that override these must call the base with a value that matches the feedback definitions in the driver JSON. |
| Poll | Invoked at the rate defined by defined by PollingInterval in the driver JSON data if polling is enabled. There is no base implementation. Drivers may use this to send their own custom polling commands that are not considered standard commands. |
| PrepareStringThenSend | Invoked right before a command is inserted into an internal sending queue. Drivers can append any common data needed to their commands here. Call the base after complete.  This should return the base call which will set CommandPrepared to true on the CommandSet. This prevents the command from being prepared again.  If the driver needs to send the command immediately, it may call Transport.Send(…) and should return true to prevent the base class from sending the command. |
| SetUserAttribute | Invoked when the user sets a user attribute. Drivers should override this if they defined any user attributes and handle the data here. There is no base implementation. |

#### Callable Methods

Any methods not listed may not be used.

|  |  |
| --- | --- |
| Method | Description & Use |
| SendCommand | Sends the specified CommandSet through the base sending logic. This will end up calling PrepareStringThenSend. |
| Log | Logs a message through the driver’s error log system. Drivers should check the property “EnableLogging” before calling this method. |
| BuildCommand | Builds a command based on the specified StandardCommandsEnum use data in the driver JSON data. |
| FireEvent | Sends an event to the main driver class that will be then sent to applications if there was a state change. |

#### Usable Properties

A driver can reference several properties at run-time. The driver may not reference any properties that are not listed.

|  |  |
| --- | --- |
| Property | Use |
| IsConnected | This value is set by the protocol when connections/disconnections happen.  Drivers must throw a connection event when this value is changed. |
| PowerIsOn | The value is set by the protocol when a power state is received from the device.  Driver must throw a power event this value is changed. By default, DeconstructPower will set this value and throw the power event. |
| WarmingUp | This value is set internally when a Power command is sent with the callback “WarmUp()”. An internal timer keeps track of the warm up period (defined in the driver’s JSON data) and will set this value to false once warm up is complete. |
| CoolingDown | Same as WarmingUp, but affects this property. |
| TimeBetweenCommands | Specifies how long to space commands out being sent to the transport.  This value may not be lower than 250ms. If it is lower, then the framework will use 250ms. |
| PollingEnabled | This value is initially set by the value found in the driver’s JSON data:  CrestronSerialDeviceApi.Api.Communication. EnableAutoPolling  This value may be changed at run-time to enable/disabling polling. |
| WaitForResponse | This values is initially set by the value found in the driver’s JSON data:  CrestronSerialDeviceApi.Api.Communication. WaitForResponse  This value must be set to true only if the driver’s responses do not have unique headers. An example of this is when polled for power the device responds with an “On” or “Off” but the response does not indicate what is on or off. |
| PendingRequest | This is set to the CommandSet that was last sent to the transport.  When WaitForResponse is true, this may be used to determine the last sent command. The driver may not set this value. |
| PollingInterval | The number of milliseconds to wait before starting to send the current polling sequence to the device.  This value is 4000ms by default. |
| PowerOnPollingSequence\* | A set of standard commands that are considered polling commands that should be set to the device at the interval of PollingInterval when PowerIsOn is true. |
| PowerOffPollingSequence\* | A set of standard commands that are considered polling commands that should be set to the device at the interval of PollingInterval when PowerIsOn is false. |
| EnableLogging | Only applications should set this value. Determines if any logging is possible. |
| EnableTxDebug | Only applications should set this value. Determines if TX debugging should happen. |
| EnableRxDebug | Only applications should set this value. Determines if RX debugging should happen. |

\*Security system device types do not use the PowerOnPollingSequence or PowerOffPollingSequence properties but instead function with the PollingHandler object in the secuiryt system SDK.

#### Security System Usable Objects

##### Polling Handler

|  |  |
| --- | --- |
| Method | Use |
| Dispose | Disposes the instance of the polling handler object. |
| AddPollingSequenceItem | While the polling handler is inactive or active this method can be called to add a PollingSequenceItem to the list of PollingSequenceItems. |
| SetPollingSequenceItemModifier | The polling modifier is used for certain standard commands that require additional properties to be filled in, like for instance the password in the login command or the zone number for request zone information. |
| SetPollingSequenceItemResource | Sets the resource property of the PollingSequenceItem with the name specified equal to the resource passed in the method. |
| GetPollingState | Returns a Boolean value representing if the PollingHandler is actively polling or idle. |
| Restart | Resets the handler to the first PollingSequenceItem in the list and enables the connection to the timer to trigger the polling commands. |
| SetPollingState | Start or stop the polling handler. |
| SetInterval | Set the interval of time between the commands that the PollingHandler sends to the driver transport. If the PollingHandler object is not told to advance to the next PollingSequenceItem the PollingHandler object will instruct the driver to send the previous polling command again. |
| AdvanceToNextPollingItem | Instructs the polling handler to move to the next PollingSequenceItem the next time the PollingHandler object triggers a command to be sent. |
| SetPollingItem | Instructs the polling handler object to go to a specific item in the PollingSequenceItems list and the next time it sends a command send that one. |
| GetPollingItem | Returns the position in the PollingSequenceItems list as a int. If a standard command is passed as a parameter the function will return the first position found in the PollingSequienceItems for that particular standard command. |

##### Dynamic Feature

The DynamicFeature class is used in the Security System SDK to provide a list of features generated at run time. This enables a driver to extend the capabilities of the SDK without having to release an updated version of the SDK in order to have the SIMPL# driver library to pass data to the SIMPL+ wrapper in a way that enables a SIMPL Windows module or SIMPL# Pro program in a manner that enables the feature to look and feel native to the user.

|  |  |
| --- | --- |
| Property | Use |
| Name | The feature’s name used to identify it. |
| Value | The value of the feature as a string. |
| PossibleValues | A list of strings that are the possible values of the feature |
| SetPollingSequenceItemResource | Sets the resource property of the PollingSequenceItem with the name specified equal to the resource passed in the method. |

#### Events and State Changes

The abstract protocol class will handle the majority of all events sent to applications. The main driver class will only send events to applications if there was a value change.

Driver may fire events to update feedback values available in the main class for applications. All events are thrown by each Deconstruct<Feature> method.

The following are available events based on the device types. Certain events will be handled internally so drivers should only throw events that the tables specify parameters on:

##### Displays

|  |  |  |
| --- | --- | --- |
| Event | Parameters | Values Changed |
| Power | Crestron.RAD.DeviceTypes.Display.Power | WarmingUp, CoolingDown, or PowerIsOn |
| Input | Crestron.RAD.DeviceTypes.Display.Input | InputIs, VideoInputSource, or AudioInputSource |
| Audio | Crestron.RAD.DeviceTypes.Display.Audio | Muted or VolumePercent |
| Connection | Crestron.RAD.DeviceTypes.Display.Connection | IsConnected |
| LampHours | Crestron.RAD.DeviceTypes.Display.Projector | LampHours |
| EnergyStar | Crestron.RAD.DeviceTypes.Display.EnergyStar | EnergyStarIsOn |
| VideoMute | Crestron.RAD.DeviceTypes.Display.VideoMute | VideoMuteIsOn |
| Volume | Handled internally with the Audio event | VolumePercent |
| Mute | Handled internally with the Audio event | Muted |
| Authentication | Boolean where true is authenticated | IsAuthenticated |
| AudioInput | Handled internally with the Input event | AudioInputSource |
| VideoInput | Handled internally with the Input event | VideoInputSource |
| WarmingUp | Handled internally with the Power event | WarmingUp |
| WarmedUp | Handled internally with the Power event | WarmingUp |
| CoolingDown | Handled internally with the Power event | CoolingDown |
| CooledDown | Handled internally with the Power event | CoolingDown |
| PoweredOn | Handled internally with the Power event | PowerIsOn |
| PoweredOff | Handled internally with the Power event | PowerIsOn |
| ActiveMediaService | ID (string) of media service that is active | ActiveMediaService |
| ActiveMediaServicePlaybackState | MediaServiceStates (enum) | ActiveMediaService.State |
| Channel | Crestron.RAD.DeviceTypes.Display. Channel | ChannelIs |

##### Cable Boxes

|  |  |  |
| --- | --- | --- |
| Event | Parameters | Values Changed |
| Power | Crestron.RAD.DeviceTypes.CableBox.Power | PowerIsOn, WarmingUp, or CoolingDown |
| Channel | Crestron.RAD.DeviceTypes.CableBox.Channel | Channelis |
| Audio | Crestron.RAD.DeviceTypes.CableBox.Audio | Muted or VolumePercent |
| Connection | Crestron.RAD.DeviceTypes.CableBox.Connection | IsConnected |
| EnergyStar | Crestron.RAD.DeviceTypes.CableBox.EnergyStar | EnergyStarIsOn |
| Volume | Handled internally with the Audio event | VolumePercent |
| Mute | Handled internally with the Audio event | Muted |
| Authentication | Boolean where true is authenticated | IsAuthenticated |
| WarmingUp | Handled internally with the Power event | WarmingUp |
| WarmedUp | Handled internally with the Power event | WarmingUp |
| CoolingDown | Handled internally with the Power event | CoolingDown |
| CooledDown | Handled internally with the Power event | CoolingDown |
| PoweredOn | Handled internally with the Power event | PowerIsOn |
| PoweredOff | Handled internally with the Power event | PowerIsOn |

##### Blu-Ray Players

|  |  |  |
| --- | --- | --- |
| Event | Parameters | Values Changed |
| Power | Crestron.RAD.DeviceTypes.  BlurayPlayer.Power | PowerIsOn, WarmingUp, or CoolingDown |
| PlaybackStatus | Crestron.RAD.DeviceTypes.  BlurayPlayer.Playback | PlayBackStatus |
| MediaInfo | Crestron.RAD.DeviceTypes.  BlurayPlayer.MediaInfoFeedback | ChapterFeedback,  TrackFeedback,  TrackElapsedTime,  TotalRemainingTime,  ChapterElapsedTime,  ChapterRemainingTime,  TotalElapsedTime,  Or TotalRemainingTime |
| Connection | Crestron.RAD.DeviceTypes. BlurayPlayer.Connection | IsConnected |
| EnergyStar | Crestron.RAD.DeviceTypes. BlurayPlayer.EnergyStar | EnergyStarIsOn |
| Authentication | Boolean where true is authenticated | IsAuthenticated |
| WarmingUp | Handled internally with the Power event | WarmingUp |
| WarmedUp | Handled internally with the Power event | WarmingUp |
| CoolingDown | Handled internally with the Power event | CoolingDown |
| CooledDown | Handled internally with the Power event | CoolingDown |
| PoweredOn | Handled internally with the Power event | PowerIsOn |
| PoweredOff | Handled internally with the Power event | PowerIsOn |

##### AV Receivers

|  |  |  |
| --- | --- | --- |
| Event | Parameters | Values Changed |
| Power | Crestron.RAD.DeviceTypes. RADAVReceiver.Power | PowerIsOn, WarmingUp, or CoolingDown |
| Input | Crestron.RAD.DeviceTypes. RADAVReceiver.Input | VideoInputSource |
| Audio | Crestron.RAD.DeviceTypes. RADAVReceiver.Audio | TreblePercentIs,  BassPercentIs, TrebleDbIs, BassDbIs,  LoudnessIsOn, or ToneControlIsOn |
| Connection | Crestron.RAD.DeviceTypes. RADAVReceiver.Connection | IsConnected |
| Volume | Crestron.RAD.DeviceTypes. RADAVReceiver.Volume | VolumePercent |
| Mute | Handled internally with the Volume event | Muted |
| Tuner | Crestron.RAD.DeviceTypes. RADAVReceiver.Connection | Tuner |
| Surround | Crestron.RAD.DeviceTypes. RADAVReceiver. SurroundMode | SurroundMode |
| Authentication | Boolean where true is authenticated | IsAuthenticated |
| AudioInput | Crestron.RAD.DeviceTypes. RADAVReceiver.AudioInput | AudioInputIs |
| VideoInput | Handled internally with the Input event | VideoInputSource |
| WarmingUp | Handled internally with the Power event | WarmingUp |
| WarmedUp | Handled internally with the Power event | WarmingUp |
| CoolingDown | Handled internally with the Power event | CoolingDown |
| CooledDown | Handled internally with the Power event | CoolingDown |
| PoweredOn | Handled internally with the Power event | PowerIsOn |
| PoweredOff | Handled internally with the Power event | PowerIsOn |

##### Video Servers

|  |  |  |
| --- | --- | --- |
| Event | Parameters | Values Changed |
| Power | Crestron.RAD.DeviceTypes.VideoServer.Power | PowerIsOn, WarmingUp, or CoolingDown |
| Connection | Crestron.RAD.DeviceTypes. VideoServer.Connection | IsConnected |
| Authentication | Boolean where true is authenticated | IsAuthenticated |
| WarmingUp | Handled internally with the Power event | WarmingUp |
| WarmedUp | Handled internally with the Power event | WarmingUp |
| CoolingDown | Handled internally with the Power event | CoolingDown |
| CooledDown | Handled internally with the Power event | CoolingDown |
| PoweredOn | Handled internally with the Power event | PowerIsOn |
| PoweredOff | Handled internally with the Power event | PowerIsOn |

##### Security Systems

|  |  |  |
| --- | --- | --- |
| Event | Parameters | Values Changed |
| Message | Crestron.RAD.DeviceTypes.SecuritySystem. MessageState | The most recent message title and content are assigned to the strings MessageTitle and MessageContent respectively. |
| Login | Crestron.RAD.DeviceTypes.SecuritySystem. AuthorizedState | IsAuthorized |
| Connection | Crestron.RAD.DeviceTypes.SecuritySystem. Connection | IsConnected |
| SoftwareVersion | Crestron.RAD.DeviceTypes.SecuritySystem. SoftwareVersion | String objects named Version, VersionModifier, PanelModel, VersionDate, RemoteVersion |
| SoftwareVersionDifference | Crestron.RAD.DeviceTypes.SecuritySystem. SoftwareVersionDifferenceState | IsCompatible |
| SystemFeatures | Crestron.RAD.DeviceTypes.SecuritySystem. SystemFeatures | Features |
| ResourceStatus | List<Crestron.RAD.Common.Resource> | Resources |
| BypassResourceStatus | List<Crestron.RAD.Common.Resource> | Resources |
| ReadyResourceStatus | List<Crestron.RAD.Common.Resource> | Resources |
| SystemStatus | Crestron.RAD.DeviceTypes.SecuritySystem.SystemStatus | String objects ACPowerStatus, BatteryStatus, TelephoneLine1Status, TelephoneLine2Status, TamperStatus, PrinterStatus, WirelessStatus |
| ArmModeButton | Crestron.RAD.DeviceTypes.SecuritySystem. ArmModeButtonsState | ArmButtonGroups |

##### Command Sending

The protocol class is responsible for formatting and sending commands to the transport class. These commands can be sent either as a CommandSet object or as a custom command.

The framework has a built-in sending queue where all commands sent will be added to an internal sending queue. This queue is maintained once every 250ms and it will remove commands that cannot be sent due to certain device states. There is a method to skip this queue and send the data at any rate detailed later in this section.  
  
 Here is a table that explains what is sent and when:

|  |  |
| --- | --- |
| Command | Behavior |
| Power Poll | Always sent |
| Power On/Off | Sent when device is not warming up or cooling down.  If a PowerOn command enters the queue while the device is cooling down then the command will remain in the queue until the device finishes cooling down. It will be sent after cooling down is complete.  The same applies to PowerOff with reverse conditions.  PowerOn/Off will automatically re-enter the queue up to 3 times if feedback from the device does not match the desired state. |
| Audio/Video Input | Sent when device is not warming up, cooling down, is connected and powered on.  If this command enters the queue while warming up then it will remain in the queue until the device is done warming up. It will be sent after warming up is complete. |
| All other commands | Will not be sent unless the device is powered on, connected, and not warming up or cooling down. The one exception to this is the Security System device type which doesn’t have a power state and will always send commands as long as the device is connected.  When the command cannot be sent due to the conditions at the time then it will be removed from the queue. |

There are three ways to send a command:

* Calling the method SendCommand(CommandSet commandSet) with a populated CommandSet object
  + This will result in the command being added to an internal sending queue
* Calling the method SendCustomCommandByName(string) where the name matches on of the defined custom commands in the following class in the driver JSON data: CrestronSerialDeviceApi.Api.CustomCommands
  + This will result in the command being added to an internal sending queue
* Calling the method Transport.Send(string, object[]) where the string is the data that needs to be transmitted and the object array may be used by a custom transport. No pre-built transports use the second parameter.
  + This will result in the command being sent immediately to the transport and skipping the queue.
  + This is useful when the driver needs to send a message to the device immediately or when a non-standard command should not be considered a custom command for use by applications.

##### CommandSet

The class CommandSet represents a command that will be sent to the transport.

|  |  |
| --- | --- |
| Property | Description & Use |
| CommandName | Friendly name of the command. |
| StandardCommand | The StandardCommandsEnum value that matches this command’s function.   The value “NotAStandardCommand” has to be used when the command is not a standard command. |
| CommandPriority | The priority of the command to determine when it has to be sent.   Section 1.4.2.6 explains how this value is used. |
| CallBack | The action that should be invoked once the command exits the sending queue and is sent to the device.  An example of the use of this is PowerOn needs to start a warm up sequence when it is sent to the transport, so the callback will be set to the method WarmUp. |
| Command | The string that should be sent to the device. |
| CommandGroup | The command group this command belongs to.  An example of this would be a PowerOn command would belong to the command group Power. |
| IsPollingCommand | Outputs true if the specified StandardCommand property is a polling command. |
| CommandPrepared | Specifies if the command is prepared and ready to be sent to the device without any more modifications.  The base implementation of PrepareStringThenSend will set this value to true before sending it to the internal sending queue. |
| Parameters | Parameters are discussed later. |

These are built by all the component interface methods located in this abstract class. The general pattern they use to build these commands is:

|  |
| --- |
| public virtual void MuteOn()  {  CommandSet command = BuildCommand(StandardCommandsEnum.MuteOn, CommonCommandGroupType.Mute, CommandPriority.Normal, "Mute On");  if (command != null)  {  SendCommand(command);  }  } |

The method BuildCommand will retrieve the command string (yellow) found in the driver JSON data based on the StandardCommandsEnum value (green). The property “StandardCommand” needs to be the value of the StandardCommandsEnum defined.

In this example, StandardCommandsEnum.MuteOn is equal to 4602.

|  |
| --- |
| "MuteOn": {  "StandardCommand": 4602,  "Command": "\\u0000\\u0003\\u0002\\u0001",  "Parameters": []  }, |

Parameters are defined as:

|  |  |
| --- | --- |
| Property | Description & Use |
| Id | The ID of the parmater. This should match the string in “!$[ParameterId]” where Id would be “ParameterId”. |
| Max | The max value the parameter can be |
| Min | The minimum value of the parameter |
| Type | The type of conversion to use when sending this parameter to the device:  0 – Send as string with no modifications  1 – Convert from ASCII to Hex equivalent. “10” would become “0x31 0x30”  2 – Convert from decimal value to hex equivalent. “10” would become 0x0A |
| StaticDataWidth | The total number of characters the parameter should be padded to. Padding will not be done if the parameter exceeds this value. |
| PadCharacter | The character to use in padding |
| PadDirection | The direction used to pad the parameter |

Parameters are denoted in commands in JSON using the “!$[ParameterId]” tag. By default, parameters are used in the framework for the parameter “id” and for volume commands. They can be implemented by a driver and replaced within PrepareStringThenSend. Parameters are not required to be used – drivers may handle their define their own syntax for parameters within commands as long as they handle them.

##### Custom Commands

Custom commands are defined by the component interface ICustomCommandCollection and all the abstract device type classes implement this interface. A driver developer may not override any methods that are part of this interface.

ICustomCommandCollection allows driver developers to add custom commands for applications to use. This is useful when there is a manufacturer-specific command that is not considered a standard command by the framework that applications should be able to use. An example of a custom command that must be provided to applications would be “XFinity” since it is not a standard command but it is a proprietary command frequently seen on a cable TV provider’s remote.

Any commands added to this collection will be visible to applications so internal API calls should not be added as custom commands.

All custom commands are stored in the driver JSON data and will be explained later in the Driver JSON Data section.

While it is possible to add custom commands dynamically after a driver is initialized, it is best practice to include this data in the driver JSON data so that it is available before the driver is instantiated by an application.

##### Sending Directly to the Transport Class

Data can also be sent to the device by calling the Send method on the transport class. This will skip the internal sending queue and the transport will send the data immediately to the device.

This is useful when drivers need to send some custom configuration to the device, such as setting the device to use the COM port for control, or when authenticating with the device. Neither of these commands would be something a driver developer would expose to applications using custom commands so they have the option of sending them this way. Examples of both of these use-cases and some others are shown in the Writing a Driver section.

Use of this method of sending should be limited since it will skip the internal sending queue. Most devices do not support processing commands faster than about 100-250ms per message so sending too many commands this way may cause the device to react unpredictably.

##### Polling Commands

Polling commands can be added to the properties PowerOnPollingSequence and PowerOffPollingSequence.

The PowerOffPollingSequence only allows PowerPoll to be added.  
  
Commands specified in these properties will remain the queue forever. They will have the lowest priority and they will be sent once every PollingInterval. Polling commands sent at run-time using SendCommand will not be enqueued if there already is a polling command with the same standard command in the queue.

Drivers may override the method Poll() which has no logic in the default implementation. This method is called once every polling cycle and the driver can send any polling commands here that are not part of the polling sequence.

##### Security System Polling Commands

Since security systems do not have a power state there no hardwired condition to not poll unless the device is not connected, or the driver programs a condition to stop the polling.

Security System device polling is handled through instances of the Polling Hander object. Multiple instances of the PollingHandler class can be utilized at the same time with their own sequence of polling commands.

However, all instances of the PollingHandler object will send their sequenced polling commands through the transport queuing which effectively means the different instances will take turns sending commands like a zipper.

Since each instance of the PollingHandler class can have different command intervals it is possible to have some higher priority commands sent in a faster queue while lower priority commands that need to be sent less often can be sent in a slower queue.

#### ValidatedRxData

The class ValidatedRxData defines data that has been validated completely or partially by the driver’s Response Validation class. This class is used by the protocol class and will be used by a driver developer. How the framework uses it is explained on the next page.

|  |  |
| --- | --- |
| Property | Description & Use |
| Ready | Specifies if the response has been fully received and is ready to be parsed by the protocol class.  Once a message is processed that was marked as Ready the internal RX buffer will clear itself. |
| Ignore | Specifies if the response should be ignored and the internal RX buffer should be cleared. |
| Data | The received data from the device.  This should not be the entire message from the device, only the relevant part of the response. For example, if this was a response to PowerPoll, the CommandGroup would be set to Power and the Data contents would be “\x01” if the driver JSON data defined Power On feedback as “\x01”. |
| CommandGroup | The command group of the Data belongs to. |
| CustomCommandGroup | If there are non-standard responses that need to be handled by the driver internally, the driver’s response validation class may set this to a unique value and override ChooseDeconstructMethod in their protocol class to handle the response.  The framework does not reference this value. |

#### Response Handling

When the transport class receives data, it will call the method DataHandler within the protocol class. DataHandler will add the received response into an internal blocking queue. The driver may not override this method. Received data is removed from the queue when not blocked and will do the following:

1. Invoke RxOut (defined in IBasicLogger) with the received bytes
2. Append the received bytes to a string buffer
3. Call to ResponseValidation.ValidateResponse with the *string buffer contents* and the last command group that was sent.
   1. Exceptions thrown by the drivers implementation of ValidateResponse will be caught, logged, and the RX buffer will be cleared
4. The connection state will be updated to true and a Connection event will be made if the current state is disconnected
5. If the returned validated data should be ignored the RX buffer will be cleared and this method will end here
6. If the returned validated data is not ready or Data is empty then an internal counter will be increment to keep track of partial or unrecognized responses and the method will end here. This will clear once the data is ready and not empty.
7. The last command group value will be set to the command group of the validated data if it is not unknown. Otherwise, if WaitForResponse is true then the validated data’s command group will be assigned to the last command group.
8. If this was a response to a pending polling command then it will be marked as having received a response preventing it from timing out.
9. ChooseDeconstructMethod is now called with the validated data object.

#### User Attributes

User attributes are used for getting device-specific data to the driver and for relaying any setup messages required for getting the device to work with the driver.

These must be defined in the driver’s JSON data:

CrestronSerialDeviceApi.UserAttributes

The driver’s protocol class must override the appropriate SetUserAttribute methods to use the values supplied by the application. There is no default implementation in the protocol class. The main class will send all received values to the protocol using the SetUserAttribute method. These values are stored internally and then sent to the protocol when it is instantiated If the protocol does not exist when the applications calls these methods.

An attribute is defined as follows:

|  |  |
| --- | --- |
| Property | Use |
| TypeName | Specifies the type of user attribute.  The following are allowed:  -DeviceId  -OnScreenId  -MessageBox  -Custom  “DeviceId” should be used when an ID is required for communication with the device. For example, the NEC display API requires an ID value in all commands sent to the device and this ID comes from the configuration screen on the display.  “OnScreenId” must be used when a value is available on the device that is required for communication with the device. For example, LG’s 1018 API requires a pass code to be generated and provided to the driver for communication.  “MessageBox” must be used to relay messages to users using an application. For example, if a driver requires a certain setting to be set before communication is possible then this type would be used and the description would describe the necessary steps.  “Custom” must be used for all other user attribute types. For example, you would use this for a MAC address of the device if WakeOnLAN is supported and there is no other way to turn the device on. |
| ParameterId | The ID used by applications when making a call to SetUserAttribute(). |
| Label | A friendly name for the user attribute. |
| Description | A description of the user attribute. |
| Persistent | Specifies whether the attribute is considered non-changing, such as the serial number of a device.  Persistent values must be stored by the application. |
| RequiredForConnection | Specifies if the connection state relies on this attribute being sent.  One of the following values must be used:  -None  -Before  -After  “None” specifies that the attribute is not required for communication.  “Before” specifies that the attribute is required before a connection is possible.  “After” specifies that the attribute is required after connecting to the device. |
| Data | Represents information about the type of value that is needed for the user attribute. |
| Data.DataType | Specifies the data type of the attribute.  This must be one of the following values: -String  -Number  -Boolean  -Hex |
| Data.Mask | Not used |
| Data.DefaultValue | The default value this attribute will use, if any. |

#### Authentication

Some devices will require a username and/or a password before allowing the driver full control.

Authentication details are stored within the driver’s JSON data:

CrestronSerialDeviceApi.Api.Communication.Authentication

Drivers that require authentication need to support IAuthentication and IAuthentication2. IAuthentication supports usernames and passwords using CrestronDataStore. IAuthentication2 supports them using two methods: OverrideUsername and OverridePassword.

Drivers should first use the username/password found in CrestronDataStore if they exist. If they do not exist then the driver must use the values provided to the driver using OverrideUsername and OverridePassword.

When a driver authenticates with a device, the driver needs to throw an Authentication event.

Drivers are not responsible for storing any data in CrestronDataStore but they are responsible for retrieving data found in it if a key is provided by the application

##### IAuthentication

To support this interface the driver’s abstract class needs to override the methods UsernameKey and PasswordKey and pass these values to the driver’s protocol implementation. Applications will call these methods with the key that corresponds to a value they inserted in the CrestronDataStore. Drivers should attempt to use the key to retrieve the username/password from CrestronDataStore.

##### IAuthentication2

To support this interface the driver’s abstract class needs to override the methods OverrideUsername and OverridePassword and pass these values to the driver’s protocol implementation.

#### One Way Serial Drivers

The framework supports one-way serial drivers. All command sending logic will check if the driver supports power feedback before checking PowerIsOn.

Older versions of the framework did not have this logic implemented so drivers should set PowerIsOn to true within their protocol’s constructor if the device provides no power feedback.

### Driver JSON Data

The main source of all information about a driver comes from the driver’s JSON data that is an embedded resource in the driver. All drivers must create and use this file.

While each device type has its own definition of the main class, the object remains the same between all device types.

The next several sections will go into detail about each node, how they are used, and any limitations that exist.

#### CrestronSerialDeviceApi

This class contains several properties:

* DeviceSupport
* DeviceSupport2
* GeneralInformation
* API
* UserAttributes

##### DeviceSupport

This is a dictionary where the key is the type “SupportedFeatureEnum” defined by each device type and the value is a Boolean indicating if there is support for the feature.

The values of SupportedFeatureEnum match the values found on CommonFeatureSupportEnum. The reason each device type has its own set is to limit what features a device type could have.

All standard features supported by a driver must go into this dictionary except for ones that are not allowed based on limitations. Any values not allowed in this dictionary must go into the DeviceSupport2 list.

The limitations on what values can be inserted into this dictionary differ based on the device type. The next sections cover these limitations. If these limitations are not followed, then the driver will fail to load on older applications such as Pyng and AVF.

An example of this property in JSON:

|  |
| --- |
| "DeviceSupport": {  "SupportsDiscretePower": true,  "SupportsPowerFeedback": true,  "SupportsDiscreteMute": true,  "SupportsChangeVolume": true,  "SupportsSetVolume": true,  "SupportsSetInputSource": true,  "SupportsWarmUpTime": true,  "SupportsCoolDownTime": true,  "SupportsDisconnect": true,  "SupportsReconnect": true  }, |

##### Allowed Display DeviceSupport Values

The following values may only exist in this dictionary:

|  |
| --- |
| SupportsChangeVolume |
| SupportsCoolDownTime |
| SupportsDisconnect |
| SupportsDiscreteMute |
| SupportsDiscretePower |
| SupportsMute |
| SupportsMuteFeedback |
| SupportsPowerFeedback |
| SupportsReconnect |
| SupportsSetVolume |
| SupportsTogglePower |
| SupportsVolumePercentFeedback |
| SupportsWarmUpTime |
| SupportsFeedback |
| SupportsInputFeedback |
| SupportsLampHours |
| SupportsSetInputSource |
| SupportsToggleVideoMute |
| SupportsToggleEnergyStar |
| SupportsDiscreteEnergyStar |
| SupportsEnergyStarFeedback |
| SupportsDiscreteVideoMute |
| SupportsVideoMuteFeedback |
| SupportsOnScreenDisplayFeedback |

##### Allowed Blu-ray Player DeviceSupport Values

|  |
| --- |
| SupportsArrowKeys |
| SupportsAsterisk |
| SupportsBack |
| SupportsClear |
| SupportsColorButtons |
| SupportsCoolDownTime |
| SupportsDash |
| SupportsDisconnect |
| SupportsDiscretePower |
| SupportsEnter |
| SupportsExit |
| SupportsForwardScan |
| SupportsForwardSkip |
| SupportsHome |
| SupportsKeypadBackSpace |
| SupportsKeypadNumber |
| SupportsMenu |
| SupportsPause |
| SupportsPeriod |
| SupportsPlay |
| SupportsPound |
| SupportsPowerFeedback |
| SupportsReconnect |
| SupportsRepeat |
| SupportsOptions |
| SupportsReturn |
| SupportsReverseScan |
| SupportsReverseSkip |
| SupportsStop |
| SupportsTogglePower |
| SupportsWarmUpTime |
| SupportsFeedback |
| SupportsPlayBackStatusFeedback |
| SupportsTrackFeedback |
| SupportsChapterFeedback |
| SupportsTrackElapsedTimeFeedback |
| SupportsTrackRemainingTimeFeedback |
| SupportsChapterElapsedTimeFeedback |
| SupportsChapterRemainingTimeFeedback |
| SupportsTotalElapsedTimeFeedback |
| SupportsTotalRemainingTimeFeedback |
| SupportsAudio |
| SupportsDisplay |
| SupportsEject |
| SupportsSubtitle |
| SupportsToggleEnergyStar |
| SupportsDiscreteEnergyStar |
| SupportsEnergyStarFeedback |
| SupportsPopUpMenu |
| SupportsTopMenu |
| SupportsSelect |

##### Allowed Cable Box DeviceSupport Values

|  |
| --- |
| SupportsArrowKeys |
| SupportsAsterisk |
| SupportsBack |
| SupportsChangeChannel |
| SupportsChangeVolume |
| SupportsChannelFeedback |
| SupportsClear |
| SupportsColorButtons |
| SupportsCoolDownTime |
| SupportsDash |
| SupportsDisconnect |
| SupportsDiscreteMute |
| SupportsDiscretePower |
| SupportsDvrCommand |
| SupportsEnter |
| SupportsExit |
| SupportsFavorite |
| SupportsForwardScan |
| SupportsForwardSkip |
| SupportsGuide |
| SupportsHome |
| SupportsInfo |
| SupportsKeypadBackSpace |
| SupportsKeypadNumber |
| SupportsLast |
| SupportsLetterKeys |
| SupportsLive |
| SupportsMenu |
| SupportsMute |
| SupportsMuteFeedback |
| SupportsPageChange |
| SupportsPause |
| SupportsPeriod |
| SupportsPlay |
| SupportsPound |
| SupportsPowerFeedback |
| SupportsReconnect |
| SupportsRecord |
| SupportsRepeat |
| SupportsReplay |
| SupportsReturn |
| SupportsReverseScan |
| SupportsReverseSkip |
| SupportsSetChannel |
| SupportsSetVolume |
| SupportsSpeedSlow |
| SupportsStop |
| SupportsThumbsDown |
| SupportsThumbsUp |
| SupportsTogglePower |
| SupportsVolumePercentFeedback |
| SupportsWarmUpTime |
| SupportsFeedback |
| SupportsToggleEnergyStar |
| SupportsDiscreteEnergyStar |
| SupportsEnergyStarFeedback |
| SupportsSelect |

##### Allowed Video Server DeviceSupport Values

|  |
| --- |
| SupportsArrowKeys |
| SupportsAsterisk |
| SupportsBack |
| SupportsClear |
| SupportsDash |
| SupportsDisconnect |
| SupportsEnter |
| SupportsExit |
| SupportsForwardScan |
| SupportsForwardSkip |
| SupportsHome |
| SupportsKeypadBackSpace |
| SupportsKeypadNumber |
| SupportsMenu |
| SupportsPause |
| SupportsPeriod |
| SupportsPlay |
| SupportsPound |
| SupportsReconnect |
| SupportsRepeat |
| SupportsReturn |
| SupportsReverseScan |
| SupportsReverseSkip |
| SupportsStop |
| SupportsSelect |

##### Allowed AV Receiver DeviceSupport Values

|  |
| --- |
| SupportsArrowKeys |
| SupportsChangeVolume |
| SupportsClear |
| SupportsCoolDownTime |
| SupportsDisconnect |
| SupportsDiscreteMute |
| SupportsDiscretePower |
| SupportsEnter |
| SupportsExit |
| SupportsForwardScan |
| SupportsForwardSkip |
| SupportsHome |
| SupportsMenu |
| SupportsMute |
| SupportsMuteFeedback |
| SupportsPause |
| SupportsPlay |
| SupportsPowerFeedback |
| SupportsReconnect |
| SupportsReverseScan |
| SupportsReverseSkip |
| SupportsSetVolume |
| SupportsStop |
| SupportsThumbsDown |
| SupportsThumbsUp |
| SupportsTogglePower |
| SupportsVolumePercentFeedback |
| SupportsWarmUpTime |
| SupportsFeedback |
| SupportsAudio |
| SupportsInputFeedback |
| SupportsSetInputSource |
| SupportsSelect |
| SupportsToneControlFeedback |
| SupportsDiscreteToneControl |
| SupportsToggleToneControl |
| SupportsBassFeedback |
| SupportsBassDbFeedback |
| SupportsTrebleFeedback |
| SupportsTrebleDbFeedback |
| SupportsSetBass |
| SupportsChangeBass |
| SupportsSetTreble |
| SupportsChangeTreble |
| SupportsLoudnessFeedback |
| SupportsDiscreteLoudness |
| SupportsToggleLoudness |
| SupportsSurroundModeFeedback |
| SupportsSurroundModeCycle |
| SupportsTunerFeedback |
| SupportsChangeFrequency |
| SupportsSetFrequency |
| SupportsAutoFrequency |
| SupportsFrequencyBand |
| SupportsDiscreteFrequencyBand |
| SupportsPresetRecall |
| SupportsPresetStore |
| SupportsSearch |
| SupportsSetAudioInputSource |
| SupportsSetVideoInputSource |
| SupportsAudioInputFeedback |
| SupportsVideoInputFeedback |
| SupportsSetAudioOutputSource |
| SupportsSetVideoOutputSource |
| SupportsAudioOutputFeedback |
| SupportsVideoOutputFeedback |
| SupportsSirius |
| SupportsXm |
| SupportsSiriusXm |
| SupportsHdRadio |
| SupportsInternetRadio |
| SupportsLastFm |
| SupportsPandoraRadio |
| SupportsRhapsody |
| SupportsChangePreset |
| SupportsSpotify |
| SupportsYouTube |
| SupportsYouTubeTv |
| SupportsNetflix |
| SupportsHulu |
| SupportsDirectvNow |
| SupportsAmazonVideo |
| SupportsPlaystationVue |
| SupportsSlingTv |
| SupportsAirplay |
| SupportsGoogleCast |
| SupportsDlna |
| SupportsTidal |
| SupportsDeezer |
| SupportsCrackle |
| SupportsGooglePlay |

##### Allowed Security System DeviceSupport Values

|  |
| --- |
| SupportsKeypadStatusText |
| SupportsPowerFeedback |
| SupportsUsername |
| SupportsPassword |
| SupportsKeypadEmulation |
| SupportsAsterik |
| SupportsDash |
| SupportsEnter |
| SupportsKeypadBackspace |
| SupportsKeypadNumber |
| SupportsKeypadPeriod |
| SupportsKeypadPound |
| SupportsArrowKeys |
| SupportsClear |
| SupportsExit |
| SupportsHome |
| SupportsMenu |
| SupportsSelect |
| SupportsInfo |
| SupportsLogin |
| SupportsRequestLog |
| SupportsRequestLogSetAmount |
| SupportsRequestSoftwareVersion |
| SupportsRequestSoftwareVersionDifferences |
| SupportsRequestResourceStatus |
| SupportsRequestAreaResourceStatus |
| SupportsRequestZoneResourceStatus |
| SupportsRequestOutputResourceStatus |
| SupportsRequestDoorResourceStatus |
| SupportsRequestDeviceResourceStatus |
| SupportsRequestFeatures |
| SupportsRequestSystemStatus |
| SupportsRequestAreaInfo |
| SupportsRequestZoneInfo |
| SupportsRequestOutputInfo |
| SupportsRequestDoorInfo |
| SupportsRequestDeviceInfo |
| SupportsRequestSystemSetup |
| SupportsRequestResourceName |
| SupportsRequestResourcePermission |
| SupportsBypassAll |
| SupportsBypassResource |
| SupportsUnbypassResource |
| SupportsUnbypassAll |
| SupportsResourceArmDelayed |
| SupportsResourceArmInstant |
| SupportsResourceArmAway |
| SupportsResourceArmAwayInstant |
| SupportsResourceArmAwayDelayed |
| SupportsResourceArmStay |
| SupportsResourceArmStayInstant |
| SupportsResourceArmStayDelayed |
| SupportsArmAll |
| SupportsDisarmAll |
| SupportsDisarm |
| SupportsArm |
| SupportsDisarmResource |
| SupportsRequestAreaCount |
| SupportsRequestZoneCount |
| SupportsRequestOutputCount |
| SupportsRequestDoorCount |
| SupportsRequestDeviceCount |
| SupportsSetResourceState |
| SupportsRequestBypassedResources |
| SupportsSilenceBells |
| SupportsResetSensors |
| SupportsSubscribeResourceOnOffState |
| SupportsSubscribeResourceReadyState |
| SupportsSubscribeResourceGeneralState |
| SupportsSubscribeDeviceLogging |
| SupportsSubscribeDeviceMessages |
| SupportsSubscribeDeviceConfigurationChange |
| SupportsSubscribe |
| SupportsRequestAlarm |
| SupportsRequestAreasNotReadyToArm |
| SupportsRequestUsers |
| SupportsRequestAreaAlarm |
| SupportsRequestZoneAlarm |
| SupportsRequestTamper |
| SupportsRequestAlarmMemory |
| SupportsRequestAreaAlarmMemory |
| SupportsRequestZoneAlarmMemory |
| SupportsRequestFireAlarm |
| SupportsRequestFireAlarmMemory |
| SupportsKeypadStatusText |
| SupportsPowerFeedback |
| SupportsUsername |
| SupportsPassword |
| SupportsKeypadEmulation |
| SupportsAsterik |

#### DeviceSupport2

DeviceSupport2 is of the type List<string>. All features added to this list will be internally copied to DeviceSupport if possible. Unlike DeviceSupport, any values added to this list will cause the driver to support the feature.

All features that are not allowed in DeviceSupport must be added here. Refer to Appendix G all possible values.

An example of this property in JSON:

|  |
| --- |
| "DeviceSupport2": [  "SupportsSetVideoInputSource",  "SupportsArrowKeys",  "SupportsSelect"  ], |

#### General Information

This part of the JSON will define some meta-data about the device being controlled. The following properties are available and must all be set correctly:

|  |  |  |
| --- | --- | --- |
| Property | Type | Use |
| DeviceType | String | The device type of the device.  The following values are accepted:  - Flat Panel Display  - Projector  - Video Server  - AV Receiver  - Bluray Player  - Cable Box |
| Manufacturer | String | Manufacturer of the device. |
| BaseModel | String | The model the driver was written for. |
| VersionDate | DateTime | The DateTime of when the driver was last changed. |
| DriverVersion | String | The version of the driver. This must be four octets and the first three octets must match the first three of the device type library used. The last octet may be any value. Drivers released by Crestron set this octet base on the build number on the build server. |
| SdkVersion | String | The version of the device type library used. This must be the first three octets of that library’s version. |
| Description | String | A description of the device. |
| Guid | GUID | No longer used. Use an empty GUID. |
| SupportedSeries | List<string> | List of series supported by this driver. |
| SupportedModels | List<string> | List of models supported by this driver. |

##### API

The API property in the JSON is used for the following (where applicable):

* Communication
* PowerWaitPeriod
* StandardCommands
* CustomCommands
* ArrowKeys
* LetterButtons
* ColorButtons
* InputConnections
* AudioInOut
* VideoInOut
* Feedback

##### API.Communication

**Authentication** (AuthenticationNode)

Specifies all authentication settings for the driver:

|  |  |
| --- | --- |
| Property | Use |
| UsernameRequired (bool) | Specifies if a username is required if authentication is possible. |
| PasswordRequired (bool) | Specifies if a password is required if authentication is possible. |
| Type (string) | Specifies the type of authentication required.  Values:  -None  -UsernamePassword |
| Required (bool) | Specifies if authentication is required to control the device. Only set to true if it’s not possible to control the device without authenticating. |
| UsernameMask (string) | No longer used. |
| PasswordMask (string) | No longer used. |
| DefaultUsername (string) | The default username the driver will use. |
| DefaultPassword (string) | The default password the driver will use. |

An example of this property in JSON:

|  |
| --- |
| "Authentication": {  "UsernameRequired": true,  "PasswordRequired": true,  "Type": "UsernamePassword",  "Required": false,  "UsernameMask": "",  "PasswordMask": "",  "DefaultUsername": "EPSONWEB",  "DefaultPassword": "admin"  }, |

**Baud** (eComBaudRates)

Specifies the default baud rate of the driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not specified (used by all non-COM drivers) |
| 1 | 300 |
| 2 | 600 |
| 4 | 1200 |
| 8 | 1800 |
| 16 | 2400 |
| 32 | 3600 |
| 64 | 4800 |
| 128 | 7200 |
| 256 | 9600 |
| 512 | 14400 |
| 1024 | 19200 |
| 2048 | 28800 |
| 4096 | 38400 |
| 8192 | 57600 |
| 65536 | 115200 |

**CommunicationType** (CommunicationType)

Specifies the type of communication used by the device.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not specified |
| 0 | Serial (COM) |
| 1 | IP |
| 2 | IR |
| 3 | CEC |
| 4 | Crestron Connected |
| 99 | None |

**DataBits** (eComDataBits)

Specifies the data bits confirguarion on COM drivers.

|  |  |
| --- | --- |
| JSON value | Actual value |
| 0 | Not specified |
| 7 | 7 data bits |
| 8 | 8 data bits |

**DeviceId** (int)

The default device ID used by the driver. This must be 0 if a device ID is not used.

**EnableAutoPolling** (bool)

Specifies if the driver will start polling when it is instantiated. This should be set to false if the driver doesn’t have a polling sequence or if polling should be disabled until a certain condition is met. For example, devices that require authentication need to disable polling so that they can send the username/password otherwise a polling command could be treated as the username/password by the device.

**EnableAutoReconnect** (bool)

Specifies if the driver will attempt to reconnect to a device if the connection is lost. This may only be set to true for IP transports and the best practice is to have this set to true.

**HwHandshake** (eComHardwareHandshakeType)

Specifies the hardware handshake configuration on a COM driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not Specified |
| 0 | None |
| 1 | RTS |
| 2 | CTS |
| 3 | RTS/CTS |

**IpProtocol** (ethernetProtocol)

Specifies the type of IP protocol used by the driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not Specified |
| 0 | TCP |

**IsSecure** (bool)

Specifies if the device supports authentication.

**IsUserAdjustable** (bool)

Specifies if any of the COM port configuration settings are adjustable by the user.  
The property UserAdjustableProperties will specify which specific settings are adjustable.

**Parity** (eComParityType)

Specifies the parity configuration on a COM driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not Specified |
| 0 | None |
| 1 | Even |
| 2 | Odd |

**Port** (int)

Specifies the default port an IP driver should use. The driver will not automatically use this value but applications will be able to see it.

**Protocol** (eComProtocolType)

The protocol configuration on a COM driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not Specified |
| 0 | RS-232 |
| 1 | RS-422 |
| 2 | RS-485 |

**ResponseTimeout** (uint)

The amount of time in milliseconds until a polling command is considered to time out. Once a command times out it will be resent when the polling cycle resets.

**StopBits** (eComStopBits)

The stop bits configuration on a COM driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not Specified |
| 1 | 1 stop bit |
| 2 | 2 stop bits |

**SwHandshake** (eComSoftwareHandshakeType)

The software handshake configuration on a COM driver.

|  |  |
| --- | --- |
| JSON value | Actual value |
| -1 | Not Specified |
| 0 | None |
| 1 | XON |
| 2 | XONT |
| 3 | XONR |

**TimeBetweenCommands** (uint)

The amount of time in milliseconds the driver must wait in between sending commands to the device. The minimum value is 250ms. The minimum value is used if a value is specified that is lower than the minimum value.

**UserAdjustableProperties** (List<string>)

Specifies the settings that may be adjusted by the application. This includes all COM configuration and IP configuration settings. The values must match the associated property’s name.

**WaitForResponse** (bool)

Specifies if the driver needs to wait until a valid response is received before sending the next command. This should only be set to true if responses are ambiguous and the driver cannot figure out the context of the response.

An example of this property (IP driver) in JSON:

|  |
| --- |
| "Communication": {  "CommunicationType": 1,  "Protocol": -1,  "Baud": -1,  "Parity": -1,  "HwHandshake": -1,  "SwHandshake": -1,  "DataBits": 0,  "StopBits": -1,  "Port": 9761,  "EnableAutoPolling": false,  "EnableAutoReconnect": true,  "TimeBetweenCommands": 200,  "ResponseTimeout": 4000,  "WaitForResponse": false,  "IpProtocol": 0,  "IsUserAdjustable": false,  "Authentication": {  "Type": "None",  "Required": false  },  "IsSecure": false,  "UserAdjustableProperties": [  ],  "DeviceId": 0  }, |

##### PowerWaitPeriod

Specifies the warm up and cool down settings on a driver.

**UserLocalWarmupTimer** (bool)

Set to true if the framework should handle the warm up sequence using a timer. Set to false if the device provides accurate warm up feedback and handle the feedback within the driver.

**WarmUpTime** (uint)

The amount of time the framework should use for its warm up timer.

**UseLocalCooldownTimer** (bool)

Set to true if the framework should handle the cool down sequence using a timer. Set to false if the device provides accurate cool down feedback and handle the feedback within the driver.

**CoolDownTime** (uint)

The amount of time the framework should use for its cool down timer.

Example of this property in JSON:

|  |
| --- |
| "PowerWaitPeriod": {  "UseLocalWarmupTimer": true,  "WarmUpTime": 4,  "UseLocalCooldownTimer": true,  "CoolDownTime": 4  }, |

##### StandardCommands

This is where the majority of standard commands will be defined used by the framework.

This property is of type Dictionary<StandardCommandsEnum, Commands> where the key is the standard command and the value is the information about the standard command.

Only standard commands found in Appendix A may be used in JSON. All other standard commands (Appendix B) must be supported in the protocol class instead by overriding the appropriate method and building the command within the protocol. An example of this would be (this is explained in earlier sections):

|  |
| --- |
| public override void PlayPause()  {  CommandSet command = new CommandSet("PlayPause", "keypress/play", CommonCommandGroupType.Other, null, false, CommandPriority.Normal, StandardCommandsEnum.PlayPause);  SendCommand(command);  } |

The value of type Commands is defined as:

|  |  |
| --- | --- |
| Property | Use |
| StandardCommand | The value of the StandardCommandsEnum used as the key. Appendix A contains these values. |
| Command | The command string that will be sent to the device.  Developers can insert extra characters/strings here that can be replaced in PrepareStringThenSend, which is useful for volume commands. |
| Parameters | No longer used |

An example of this property in JSON is on the next page:

|  |
| --- |
| "StandardCommands": {  "Hdmi1": {  "StandardCommand": 400,  "Command": "INPUT\_SELECT hdmi1",  },  "Hdmi2": {  "StandardCommand": 401,  "Command": "INPUT\_SELECT hdmi2",  },  "Hdmi3": {  "StandardCommand": 402,  "Command": "INPUT\_SELECT hdmi3",  },  "Hdmi4": {  "StandardCommand": 403,  "Command": "INPUT\_SELECT hdmi4",  }  }, |

##### CustomCommands

Custom commands defined by the driver will be in this property. It is of type List<CustomCommand> where CustomCommand is defined as:

|  |  |
| --- | --- |
| Property | Use |
| Command | The command string that should be sent to the device. |
| Name | The friendly name of the command that applications will be able to see. |

An example of a custom command in JSON:

|  |
| --- |
| "CustomCommands": [  {  "Name": "Photo & Video",  "Command": "APP\_LAUNCH com.webos.app.photovideo",  },  {  "Name": "Music",  "Command": "APP\_LAUNCH com.webos.app.music",  }  ], |

##### ArrowKeys

Individual arrow keys supported by the driver must be specified here. This property is of type List<ArrowDirections>.

The following device types support this feature:

* Cable Box
* Displays and Projectors
* Blu-ray players
* AV receivers
* Video servers

Valid values are:

|  |  |
| --- | --- |
| JSON Value | Actual Value |
| 0 | Up |
| 1 | Down |
| 2 | Left |
| 3 | Right |

An example of this property in JSON:

|  |
| --- |
| "ArrowKeys": [  0,  1,  2,  3  ], |

##### LetterButtons

Individual letter buttons supported by the driver must be specified here. This property is of type List<LetterButtons>.

The following device types support this feature:

* Cable Box

Valid values are:

|  |  |
| --- | --- |
| JSON Value | Actual Value |
| 0 | A |
| 1 | B |
| 2 | C |
| 3 | D |

An example of this property in JSON:

|  |
| --- |
| "LetterButtons": [  0,  1,  2,  3  ], |

##### ColorButtons

Individual letter buttons supported by the driver must be specified here. This property is of type List<LetterButtons>.

The following device types support this feature:

* Cable Box
* Blu-ray Player

Valid values are:

|  |  |
| --- | --- |
| JSON Value | Actual Value |
| 0 | Blue |
| 1 | Green |
| 2 | Red |
| 3 | Yellow |

An example of this property in JSON:

|  |
| --- |
| "ColorButtons": [  0,  1,  2,  3  ], |

##### InputConnections

This property will specify what inputs the device supports. While VideoInOut and AudioInOut have superseded this, it is still required to fill this property out. Only displays and projectors support this property. This property is of type List<InputDetail> where InputDetail is defined as:

|  |  |
| --- | --- |
| Property | Use |
| InputType | The input type like HDMI1 or VGA3 |
| InputConnector | The input connector like HDMI or VGA |
| Description | A description of the input |

Valid InputType values:

|  |  |
| --- | --- |
| JSON Value | Actual Value |
| 200 | Vga1 |
| 201 | Vga2 |
| 202 | Vga3 |
| 203 | Vga4 |
| 204 | Vga5 |
| 205 | Vga6 |
| 206 | Vga7 |
| 207 | Vga8 |
| 208 | Vga9 |
| 209 | Vga10 |
| 400 | Hdmi1 |
| 401 | Hdmi2 |
| 402 | Hdmi3 |
| 403 | Hdmi4 |
| 404 | Hdmi5 |
| 405 | Hdmi6 |
| 406 | Hdmi7 |
| 407 | Hdmi8 |
| 408 | Hdmi9 |
| 409 | Hdmi10 |
| 600 | Dvi1 |
| 601 | Dvi2 |
| 602 | Dvi3 |
| 603 | Dvi4 |
| 604 | Dvi5 |
| 605 | Dvi6 |
| 606 | Dvi7 |
| 607 | Dvi8 |
| 608 | Dvi9 |
| 609 | Dvi10 |
| 800 | Component1 |
| 801 | Component2 |
| 802 | Component3 |
| 803 | Component4 |
| 804 | Component5 |
| 805 | Component6 |
| 806 | Component7 |
| 807 | Component8 |
| 808 | Component9 |
| 809 | Component10 |
| 900 | Composite1 |
| 901 | Composite2 |
| 902 | Composite3 |
| 903 | Composite4 |
| 904 | Composite5 |
| 905 | Composite6 |
| 906 | Composite7 |
| 907 | Composite8 |
| 908 | Composite9 |
| 909 | Composite10 |
| 1000 | DisplayPort1 |
| 1001 | DisplayPort2 |
| 1002 | DisplayPort3 |
| 1003 | DisplayPort4 |
| 1004 | DisplayPort5 |
| 1005 | DisplayPort6 |
| 1006 | DisplayPort7 |
| 1007 | DisplayPort8 |
| 1008 | DisplayPort9 |
| 1009 | DisplayPort10 |
| 1200 | Usb1 |
| 1201 | Usb2 |
| 1202 | Usb3 |
| 1203 | Usb4 |
| 1204 | Usb5 |
| 1400 | Antenna1 |
| 1401 | Antenna2 |
| 1600 | Network1 |
| 1601 | Network2 |
| 1602 | Network3 |
| 1603 | Network4 |
| 1604 | Network5 |
| 1605 | Network6 |
| 1606 | Network7 |
| 1607 | Network8 |
| 1608 | Network9 |
| 1609 | Network10 |
| 1800 | Input1 |
| 1801 | Input2 |
| 1802 | Input3 |
| 1803 | Input4 |
| 1804 | Input5 |
| 1805 | Input6 |
| 1806 | Input7 |
| 1807 | Input8 |
| 1808 | Input9 |
| 1809 | Input10 |
| 1810 | Input11 |
| 1811 | Input12 |
| 1812 | Input13 |
| 1813 | Input14 |
| 1815 | Input15 |
| 32767 | Unknown |

Valid InputConnector values:

|  |  |
| --- | --- |
| JSON Value | Actual Value |
| 10 | Vga |
| 20 | Hdmi |
| 30 | Component |
| 40 | Composite |
| 50 | DisplayPort |
| 60 | Network |
| 70 | Antenna |
| 80 | Usb |
| 90 | Dvi |
| 100 | SVideo |
| 110 | Other |
| 120 | Unknown |

An example of this property in JSON:

|  |
| --- |
| "InputConnections": [  {  "InputType": 400,  "InputConnector": 20,  "Description": "HDMI1"  },  {  "InputType": 401,  "InputConnector": 20,  "Description": "HDMI2"  },  {  "InputType": 402,  "InputConnector": 20,  "Description": "HDMI3"  },  {  "InputType": 403,  "InputConnector": 20,  "Description": "HDMI4"  },  {  "InputType": 800,  "InputConnector": 30,  "Description": "Component1"  },  {  "InputType": 900,  "InputConnector": 40,  "Description": "Composite1"  }  ], |

##### AudioInOut

Specifies all controllable and uncontrollable audio inputs and outputs on the device. Pyng primarily uses this for AV routing. This property is of type AudioInOut.

AudioInOut contains two properties: Inputs and Outputs.

Inputs are of type List<AudioInputDetail> where AudioInputDetail is defined as:

|  |  |
| --- | --- |
| Property | Use |
| Connector | Connector defined by AudioConnections (Appendix C) |
| Description | A description of the input |
| FriendlyName | A friendly name applications can use. |
| Type | Connector type defined by AudioConnectionTypes (Appendix D) |

Outputs are of type List<AudioOutputDetail> where AudioOutputDetail is defined as:

|  |  |
| --- | --- |
| Property | Use |
| Connector | Connector defined by AudioConnections (Appendix C) |
| Description | A description of the input |
| FriendlyName | A friendly name applications can use. |
| Type | Connector type defined by AudioConnectionTypes (Appendix D) |

An example of this in JSON:

|  |
| --- |
| "AudioInOut": {  "Inputs": [  {  "type": 400,  "description": "HDMI1",  "connector": 10,  "friendlyName": "HDMI1"  },  {  "type": 401,  "description": "HDMI2",  "connector": 10,  "friendlyName": "HDMI2"  },  {  "type": 402,  "description": "HDMI3",  "connector": 10,  "friendlyName": "HDMI3"  },  {  "type": 403,  "description": "HDMI4",  "connector": 10,  "friendlyName": "HDMI4"  },  {  "type": 29901,  "description": "AnalogAudio1",  "connector": 40,  "friendlyName": "AnalogAudio1"  },  {  "type": 29902,  "description": "AnalogAudio2",  "connector": 40,  "friendlyName": "AnalogAudio2"  }  ],  "Outputs": [  {  "type": 29701,  "description": "OpticalAudio Out",  "connector": 20,  "friendlyName": "OpticalAudio Out"  }  ]  }, |

##### VideoInOut

Specifies all controllable and uncontrollable video inputs and outputs on the device. Pyng primarily uses this for AV routing. This property is of type VideoInOut.

VideoInOut contains two properties: Inputs and Outputs.

Inputs are of type List<VideoInputDetail> where VideoInputDetail is defined as:

|  |  |
| --- | --- |
| Property | Use |
| Connector | Connector defined by VideoConnections (Appendix E) |
| Description | A description of the input |
| FriendlyName | A friendly name applications can use. |
| Type | Connector type defined by VideoConnectionTypes (Appendix F) |

Outputs are of type List<VideoOutputDetail> where VideoOutputDetail is defined as:

|  |  |
| --- | --- |
| Property | Use |
| Connector | Connector defined by VideoConnections (Appendix E) |
| Description | A description of the input |
| FriendlyName | A friendly name applications can use. |
| Type | Connector type defined by VideoConnectionTypes (Appendix F) |

An example of this in JSON:

|  |
| --- |
| "VideoInOut": {  "Inputs": [  {  "type": 400,  "description": "HDMI1",  "connector": 20,  "friendlyName": "HDMI1"  },  {  "type": 401,  "description": "HDMI2",  "connector": 20,  "friendlyName": "HDMI2"  },  {  "type": 402,  "description": "HDMI3",  "connector": 20,  "friendlyName": "HDMI3"  },  {  "type": 403,  "description": "HDMI4",  "connector": 20,  "friendlyName": "HDMI4"  },  {  "type": 800,  "description": "Component",  "connector": 30,  "friendlyName": "Component"  },  {  "type": 900,  "description": "Composite",  "connector": 40,  "friendlyName": "Composite"  },  {  "type": 1200,  "description": "USB Photo \u0026 Video",  "connector": 80,  "friendlyName": "USB Photo \u0026 Video"  },  {  "type": 1201,  "description": "USB Music",  "connector": 80,  "friendlyName": "USB Music"  }  ],  "Outputs": [  ]  }, |

##### UserAttributes

Refer to section 1.4.3.7

An example of user attributes in JSON:

|  |
| --- |
| "UserAttributes": [  {  "TypeName": "OnScreenId",  "ParameterId": "OnScreenID",  "Label": "Passcode needed for communication",  "Description": "This is required to control the display. All Settings \u003e Network - While Network is highlighted press 828888 on the remote and generate a key code.",  "Persistent": true,  "RequiredForConnection": "Before",  "Data": {  "DataType": "String",  "Mask": "",  "DefaultValue": ""  }  },  {  "TypeName": "Custom",  "ParameterId": "MACAddress",  "Label": "MAC Address of LG 2018 Display",  "Description": "This is required to turn the display on. All Settings \u003e Network \u003e Wired/Wifi Connection \u003e MAC Address. Please provide in the format of AA:BB:CC:DD:EE:FF",  "Persistent": true,  "RequiredForConnection": "Before",  "Data": {  "DataType": "String",  "Mask": "",  "DefaultValue": ""  }  },  {  "TypeName": "MessageBox",  "ParameterId": "Configuration Notice 1",  "Label": "Setup Instructions",  "Description": "To enable IP control, Navigate to All Settings \u003e Put focus on Network icon \u003e On IR Remote enter the following sequence: 828888 \u003e Enable IP Control",  "Persistent": false,  "RequiredForConnection": "None",  "Data": {  "DataType": "",  "Mask": "",  "DefaultValue": ""  }  },  {  "TypeName": "MessageBox",  "ParameterId": "Configuration Notice 2",  "Label": "Setup Instructions",  "Description": "To enable powering on the device, Wake On LAN must be enabled on the display. Settings \u003e General \u003e Mobile TV On and set Turn on via Wi-Fi to on",  "Persistent": false,  "RequiredForConnection": "None",  "Data": {  "DataType": "",  "Mask": "",  "DefaultValue": ""  }  }  ] |

##### Media Service Providers

Specifies all controllable and uncontrollable media service applications, such as Netflix, Hulu, TV Tuner, and Spotify. Also specifies which component interfaces and their associated Support<feature> properties are supported on each media service.

|  |  |
| --- | --- |
| Property | Use |
| SupportsMediaServices | Specifies if media services are supported |
| HasDirectAccessToApps | Specifies if this driver is able to issue commands to select certain media services |
| SupportsActiveMediaServiceFeedback | Specifies if the driver supports reporting the current active media service |
| SupportsActiveMediaServiceStateFeedback | Specifies if the driver supports reporting the current active media service’ service state |
| SupportsMediaServiceSubscriptionStateFeedback | Specifies if the driver supports reporting the current active media service’s subscription state |
| Available Services | List of available media services (defined later) |
| MediaServicePollCommand | The command to send to the device that will poll it for the active media service. This does not have to be set if the InputPoll command retrieves this information |
| MediaServicePollingEnabled | Specifies if this driver should send the MediaServicePollCommand. Set this to false if InputPoll is retrieving the media service feedback |
| MediaServiceSubscriptionStatePollCommand | The command to send to the device to get the current subscription state of a media service |
| MediaServiceSubscriptionStatePollingEnabled | True if MediaServiceSubscriptionStatePollCommand should be sent to the device |
| MediaServicePlaybackStatePollCommand | The command to send to the device to get the current playback state of the active media service |
| MediaServicePlaybackStatePollingEnabled | True if MediaServicePlaybackStatePollCommand should be sent to the device |

A Media service is defined as

|  |  |
| --- | --- |
| Property | Use |
| Id | Unique ID for the media service |
| FriendlyName | A friendly name |
| Command | The command string to send to the device to select this service |
| IsBranded | True if the friendly name is a branded name |
| IsSelectable | True if this media service is selectable. If a media service is not selectable, then it cannot be supported by ActiveMediaServiceFeedback. |
| SupportedFeatureData | Defines which component interfaces are supported when this media service is in-use, and which associated Supports<feature> properties are still supported. |

Media service feedback is handled with the following properties \*only active media service feedback is supported in the framework. A driver can implement this on their own:

|  |  |
| --- | --- |
| Property | Use |
| GroupHeader | Unique ID for the media service |
| ActiveServiceFeedbackData | Contains a group header for messages specifically about the active media service, if applicable. Feedback is a Dictionary<string, string> where the key is the media service ID defined in AvailableServices, and the value is the response from the device. Drivers can override DeConstructActiveMediaServiceFeedback and modify the response to match the values specified here. |

An example of media services in JSON:

|  |
| --- |
| "MediaServiceProviders": {  "SupportsMediaServices": true,  "HasDirectAccessToApps": true,  "SupportsActiveMediaServiceFeedback": true,  "SupportsActiveMediaServiceStateFeedback": false,  "SupportsMediaServiceSubscriptionStateFeedback": false,  "AvailableServices": [  {  "Id": "amazon",  "FriendlyName": "Amazon",  "Command": "APP\_LAUNCH amazon",  "IsBranded": true,  "IsSelectable": true,  "SupportedFeatureData": [  {  "ComponentInterface": "INavigation",  "SupportStatements": [  "SupportsUpDown",  "SupportsLeftRight",  "SupportsArrowKeys",  "SupportsSelect",  "SupportsExit",  "SupportsHome"  ]  },  {  "ComponentInterface": "IMediaTransport",  "SupportStatements": [  "SupportsForwardScan",  "SupportsReverseScan",  "SupportsBack",  "SupportsReturn"  ]  },  {  "ComponentInterface": "IBasicVideoDisplay6",  "SupportStatements": [  "SupportsColorButtons"  ]  }  ]  },  {  "Id": "hulu",  "FriendlyName": "Hulu",  "Command": "APP\_LAUNCH hulu",  "IsBranded": true,  "IsSelectable": true,  "SupportedFeatureData": [  {  "ComponentInterface": "INavigation",  "SupportStatements": [  "SupportsArrowKeys",  "SupportsUpDown",  "SupportsLeftRight",  "SupportsSelect",  "SupportsExit",  "SupportsHome"  ]  },  {  "ComponentInterface": "IMediaTransport",  "SupportStatements": [  "SupportsForwardScan",  "SupportsReverseScan",  "SupportsBack",  "SupportsReturn"  ]  },  {  "ComponentInterface": "IMediaTransport2",  "SupportStatements": [  "SupportsPlayPause"  ]  },  {  "ComponentInterface": "IBasicVideoDisplay6",  "SupportStatements": [  "SupportsColorButtons"  ]  }  ]  },  {  "Id": "tvtuner",  "FriendlyName": "Live TV",  "Command": "APP\_LAUNCH com.webos.app.livetv",  "IsBranded": false,  "IsSelectable": true,  "SupportedFeatureData": [  {  "ComponentInterface": "INavigation",  "SupportStatements": [  "SupportsArrowKeys",  "SupportsUpDown",  "SupportsLeftRight",  "SupportsSelect",  "SupportsExit",  "SupportsHome"  ]  },  {  "ComponentInterface": "IKeypad",  "SupportStatements": [  "SupportsKeypadNumber",  "SupportsEnter"  ]  },  {  "ComponentInterface": "IChannel",  "SupportStatements": [  "SupportsChangeChannel",  "SupportsSetChannel"  ]  }  ]  },  {  "Id": "unknownMediaService",  "FriendlyName": "Smart TV",  "Command": "",  "IsBranded": false,  "IsSelectable": false,  "SupportedFeatureData": [  {  "ComponentInterface": "INavigation",  "SupportStatements": [  "SupportsUpDown",  "SupportsLeftRight",  "SupportsArrowKeys",  "SupportsSelect",  "SupportsExit",  "SupportsHome"  ]  },  {  "ComponentInterface": "IMediaTransport",  "SupportStatements": [  "SupportsForwardScan",  "SupportsReverseScan",  "SupportsBack",  "SupportsReturn"  ]  },  {  "ComponentInterface": "IMediaTransport2",  "SupportStatements": [  "SupportsPlayPause"  ]  },  {  "ComponentInterface": "IBasicVideoDisplay6",  "SupportStatements": [  "SupportsColorButtons"  ]  }  ]  }  ],  "FeedbackData": {  "GroupHeader": "APP:",  "ActiveServiceFeedbackData": {  "GroupHeader": null,  "Feedback": {  "amazon": "amazon",  "hulu": "hulu",  "netflix": "netflix",  "googleplay": "googleplaymovieswebos",  "slingtv": "com.movenetworks.app.sling-tv-sling-production",  "vudu": "vudu",  "youtube": "youtube.leanback.v4",  "photo&video": "com.webos.app.photovideo",  "music": "com.webos.app.music",  "browser": "com.webos.app.browser",  "tvtuner": "com.webos.app.livetv",  "unknownMediaService": "unknownMediaService"  }  },  "PlaybackStateFeedbackData": null  },  "MediaServicePollCommand": null,  "MediaServicePollingEnabled": false,  "MediaServiceSubscriptionStatePollCommand": null,  "MediaServiceSubscriptionStatePollingEnabled": false,  "MediaServicePlaybackStatePollCommand": null,  "MediaServicePlaybackStatePollingEnabled": false  }, |

##### Video Configurations

Specifies if a driver supports video configurations, which ones it does support, what command to send for each one, and what they are named.

\*Id must unique

|  |
| --- |
| "VideoConfigurations": {  "SupportsDisplayModes": true,  "DisplayModes": [  {  "Id": "4by3",  "FriendlyName": "4:3",  "Command": "ASPECT\_RATIO 4by3"  },  {  "Id": "16by9",  "FriendlyName": "16:9",  "Command": "ASPECT\_RATIO 16by9"  },  {  "Id": "setByOriginal",  "FriendlyName": "Original",  "Command": "ASPECT\_RATIO setbyoriginal"  }  ]  }, |

##### ArtworkModes

Specifies the artwork mode feature’s on/off commands and whether or not the driver supports sending the artwork mode command instead of the power command.

|  |
| --- |
| "ArtworkModes": {  "SupportsArtworkMode": false,  "SupportsArtworkModeOnPowerOff": false,  "ArtworkModeOnCommand": "Turn Artwork On\x0A\x0D",  "ArtworkModeOffCommand": "Turn Artwork Off\x0A\x0D"  }, |

##### Feedback

The feedback property is different for each device type. The base implementation of ResponseValidation will handle feedback according to the values here. All Deconstruct<Feature> methods will also refer to the driver JSON data to figure out what kind of feedback was received and what the new state is.

Displays support the following feedback:

**Header**

A header that is present in all standard feedback

**CommandAckNak**

Two string properties for ACK and NAK.

**PowerFeedback, EnergyStarFeedback, MuteFeedback, VideoMuteFeedback, and OnScreenDisplayFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are two states, On and Off, and the values are the device response that corresponds to those states.

**InputFeedback**

Contains GroupHeader (string) which stores the header for this response and a list where the values are of type InputFeedbackValues that is defined as:

|  |  |
| --- | --- |
| Property | Use |
| InputType (int) | The input type that corresponds to the VideoConnections enumerable (Appendix E). |
| Feedback (string) | The response from the device for the defined input. |

**VolumeFeedback and LampHourFeedback**

Contains GroupHeader (string) which stores the header for this response.

An example of a display’s feedback JSON:

|  |
| --- |
| "Feedback": {  "CommandAckNak": {  "Ack": "OK",  "Nak": "NG"  },  "Header": "",  "PowerFeedback": {  "Feedback": {  "On": "on",  "Off": "off"  },  "GroupHeader": ""  },  "MuteFeedback": {  "Feedback": {  "On": "on",  "Off": "off"  },  "GroupHeader": "MUTE:"  },  "InputFeedback": {  "GroupHeader": "APP:",  "Feedback": [  {  "InputType": 400,  "Feedback": "com.webos.app.hdmi1"  },  {  "InputType": 401,  "Feedback": "com.webos.app.hdmi2"  },  {  "InputType": 402,  "Feedback": "com.webos.app.hdmi3"  },  {  "InputType": 403,  "Feedback": "com.webos.app.hdmi4"  },  {  "InputType": 800,  "Feedback": "component1"  },  {  "InputType": 900,  "Feedback": "avav1"  },  {  "InputType": 1200,  "Feedback": "com.webos.app.photovideo"  },  {  "InputType": 1201,  "Feedback": "com.webos.app.music"  }  ]  },  "VolumeFeedback": {  "GroupHeader": "VOL:"  },  "EnergyStarFeedback": {  "Feedback":{ },  "GroupHeader": ""  },  "OnScreenDisplayFeedback": {  "Feedback": { },  "GroupHeader": ""  },  "VideoMuteFeedback": {  "Feedback": { },  "GroupHeader": ""  },  "LampHourFeedback": {  "GroupHeader": ""  }  }  }, |

Cable boxes support the following feedback:

**Header**

A header that is present in all standard feedback

**CommandAckNak**

Two string properties for ACK and NAK.

**PowerFeedback, EnergyStarFeedback, and MuteFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are two states, On and Off, and the values are the device response that corresponds to those states.

**ChannelFeedback and VolumeFeedback**

Contains GroupHeader (string) which stores the header for this response.

An example of a cable box’s feedback JSON:

|  |
| --- |
| "Feedback": {  "Header": "",  "PowerFeedback": {  "Feedback": {},  "GroupHeader": ""  },  "MuteFeedback": {  "Feedback": {},  "GroupHeader": ""  },  "VolumeFeedback": {  "GroupHeader": ""  },  "ChannelFeedback": {  "GroupHeader": "CH\_STATUS"  },  "CommandAckNak": {  "Ack": "",  "Nak": ""  },  "EnergyStarFeedback": {  "Feedback": {},  "GroupHeader": ""  }  }, |

Blu-ray Players support the following feedback:

**Header**

A header that is present in all standard feedback

**CommandAckNak**

Two string properties for ACK and NAK.

**PowerFeedback and EnergyStarFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are two states, On and Off, and the values are the device response that corresponds to those states.

**TrackFeedback, ChapterFeedback, TrackElapsedTimeFeedback, TrackRemainingTimeFeedback, ChapterElapsedTimeFeedback, ChapterRemainingTimeFeedback, TotalElapsedTimeFeedback, and TotalRemainingTimeFeedback**

Contains GroupHeader (string) which stores the header for this response.

An example of a Blu-ray player’s feedback JSON:

|  |
| --- |
| "Feedback": {  "CommandAckNak": {  "Ack": "",  "Nak": "ER INVALID"  },  "Header": "@",  "PowerFeedback": {  "Feedback": {  "On": "ON",  "Off": "OFF"  },  "GroupHeader": "QPW"  },  "PlayBackStatusFeedback": {  "GroupHeader": "QPL"  },  "TrackFeedback": {  "GroupHeader": "QTK"  },  "ChapterFeedback": {  "GroupHeader": "QCH"  },  "TrackElapsedTimeFeedback": {  "GroupHeader": "QTE"  },  "TrackRemainingTimeFeedback": {  "GroupHeader": "QTR"  },  "ChapterElapsedTimeFeedback": {  "GroupHeader": "QCE"  },  "ChapterRemainingTimeFeedback": {  "GroupHeader": "QCR"  },  "TotalElapsedTimeFeedback": {  "GroupHeader": "QEL"  },  "TotalRemainingTimeFeedback": {  "GroupHeader": "QRE"  },  "EnergyStarFeedback": {  "Feedback": { },  "GroupHeader": ""  }  }, |

Video servers support the following feedback:

**Header**

A header that is present in all standard feedback

**CommandAckNak**

Two string properties for ACK and NAK.

**PowerFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are two states, On and Off, and the values are the device response that corresponds to those states.

An example of a video server’s feedback JSON:

|  |
| --- |
| "Feedback": {  "Header": "",  "CommandAckNak": {  "Ack": "HTTP/1.1 200 OK",  "Nak": "HTTP NAK"  },  "PowerFeedback": {  "Feedback": {  "On": "",  "Off": ""  },  "GroupHeader": ""  }  } |

AV Receivers support the following feedback:

**Header**

A header that is present in all standard feedback

**CommandAckNak**

Two string properties for ACK and NAK.

**PowerFeedback, MuteFeedback, LoudnessFeedback, and ToneStateFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are two states, On and Off, and the values are the device response that corresponds to those states.

**InputFeedback and AudioInputFeedback**

Contains GroupHeader (string) which stores the header for this response and a list where the values are of type InputFeedbackValues that is defined as:

|  |  |
| --- | --- |
| Property | Use |
| InputType / AudioInputType (int) | The input type that corresponds to the VideoConnections/AudioConnections enumerable (Appendix E / C). |
| Feedback (string) | The response from the device for the defined input. |

**SurroundModeFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are listed below and the values are the device response that corresponds to those states.

|  |
| --- |
| JSON Value |
| Unknown |
| Movie |
| Music |
| Game |
| Direct |
| PureDirect |
| Stereo |
| Auto |
| DolbyDigital |
| DtsSurround |
| Thx |

**ToneControlFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are two states, Bass and Treble, and the values are the device response that corresponds to those states.

**TunerFeedback**

Contains GroupHeader (string) which stores the header for this response and an object called FrequencyBandFeedback that contains its own GroupHeader (string) and a dictionary where the valid keys are “Am” and “Fm” and the values are the device response that correspond to those states.

**AudioFeedback**

**VolumeFeedback**

Contains GroupHeader (string) which stores the header for this response.

An example of an AV receiver’s feedback JSON:

|  |
| --- |
| "Feedback": {  "Header": "",  "PowerFeedback": {  "Feedback": {  "On": "01",  "Off": "00"  },  "GroupHeader": "PWR"  },  "MuteFeedback": {  "Feedback": {  "On": "01",  "Off": "00"  },  "GroupHeader": "AMT"  },  "VolumeFeedback": {  "GroupHeader": "MVL"  },  "InputFeedback": {  "GroupHeader": "SLI",  "Feedback": [  {  "InputType": 400,  "Feedback": "10"  },  {  "InputType": 401,  "Feedback": "01"  },  {  "InputType": 402,  "Feedback": "11"  },  {  "InputType": 403,  "Feedback": "02"  },  {  "InputType": 404,  "Feedback": "05"  },  {  "InputType": 405,  "Feedback": "12"  },  {  "InputType": 5000,  "Feedback": "03"  }  ]  },  "AudioInputFeedback": {  "GroupHeader": "SLI",  "Feedback": [  {  "AudioInputType": 30080,  "Feedback": "24"  },  {  "AudioInputType": 30080,  "Feedback": "25"  },  {  "AudioInputType": 30080,  "Feedback": "26"  },  {  "AudioInputType": 30100,  "Feedback": "22"  },  {  "AudioInputType": 30060,  "Feedback": "23"  }  ]  },  "AudioFeedback": {  "ToneStateFeedback": {  "GroupHeader": "",  "Feedback": {}  },  "ToneControlFeedback": {  "GroupHeader": "TFR",  "IndividualGroupHeaders": {  "Bass": "B",  "Treble": "T"  }  },  "LoudnessFeedback": {  "GroupHeader": "LDM",  "Feedback": {  "On": "01",  "Off": "00"  }  }  },  "SurroundModeFeedback": {  "GroupHeader": "LMD",  "Feedback": {  "Direct": "01",  "DtsSurround": "15",  "Movie": "03",  "Music": "06",  "PureDirect": "11",  "Stereo": "00",  "Thx": "04"  }  },  "TunerFeedback": {  "GroupHeader": "TUN",  "FrequencyBandFeedback": {  "GroupHeader": "",  "Feedback": {  "Am": "Am",  "Fm": "Fm"  }  }  },  "CommandAckNak": {  "Ack": "",  "Nak": "NAK"  }  } |

Security System devices support the following feedback:

**Header**

A header that is present in all standard feedback

**CommandAckNak**

Two string properties for ACK and NAK.

**HeartbeatFeedback, SubscribeFeedback, SilenceBellsFeedback, ResetSensorsFeedback, RequestAlarmFeedback, RequestAreaAlarmFeedback, RequestZoneAlarmFeedback, RequestAlarmMemoryFeedback, RequestFireAlarmFeedback, RequestFireAlarmMemoryFeedback, RequestAreaAlarmMemoryFeedback, RequestZoneAlarmMemoryFeedback, RequestTamperFeedback, RequestBypassedResourcesFeedback, RequestSystemStatusFeedback, SystemFeaturesFeedback, SetResourceStateFeedback,** **SystemSetupFeedback, SystemStatusFeedback, AreaInfoFeedback, DoorInfoFeedback, DeviceInfoFeedback, AreaCountFeedback, ZoneCountFeedback, OutputCountFeedback, DoorCountFeedback, DeviceCountFeedback, ZoneInfoFeedback, OutputInfoFeedback, RequestSoftwareVersionFeedback, RequestInfoFeedback, BypassFeedback, UnbypassFeedback, ArmFeedback, DisarmFeedback, RequestUsersFeedback, RequestResourceNameFeedback**

Contains GroupHeader (string) which stores the header for this response and a CommandAckNak object which contains two string properties for ACK and NAK.

**LoginFeedback**

Contains GroupHeader (string) which stores the header for this response, a CommandAckNak object which contains two string properties for ACK and NAK and a dictionary where the valid keys are listed below. The values are the device response that corresponds to those states.

|  |
| --- |
| JSON Value |
| Unknown |
| Authorized |
| Unauthorized |

**RequestSoftwareVersionDifferencesFeedback**

Contains GroupHeader (string) which stores the header for this response, a CommandAckNak object which contains two string properties for ACK and NAK and a dictionary where the valid keys are listed below. The values are the device response that corresponds to those states.

|  |
| --- |
| JSON Value |
| Unknown |
| Yes |
| No |

**RequestResourceStatusFeedback, RequestAreaResourceStatusFeedback, RequestZoneResourceStatusFeedback, RequestOutputResourceStatusFeedback, RequestDoorResourceStatusFeedback, RequestDeviceResourceStatusFeedback**

Contains GroupHeader (string) which stores the header for this response, a CommandAckNak object which contains two string properties for ACK and NAK and a dictionary where the valid keys are listed below. The values are the device response that corresponds to those states.

|  |
| --- |
| JSON Value |
| Unknown |
| Unassigned |
| Armed |
| Disarmed |
| Bypassed |
| Short |
| Missing |
| Error |
| Name |

**RequestResourcePermissionAreaFeedback, RequestResourcePermissionZoneFeedback, RequestResourcePermissionDeviceFeedback, RequestResourcePermissionDeviceFeedback, RequestResourcePermissionDoorFeedback, RequestResourcePermissionOutputFeedback**

Contains GroupHeader (string) which stores the header for this response and a dictionary where the valid keys are listed below. The values are the device response that corresponds to those states.

|  |
| --- |
| JSON Value |
| Unknown |
| Authorized |
| Unauthorized |

An example of a security system device’s feedback JSON:

|  |
| --- |
| "Feedback": {  "CommandAckNak": {  "Ack": "\u00FE",  "Nak": "\u00FD"  },  "Header": "\u0001",  "LoginFeedback": {  "AckNak": {  "Ack": "\u0001\u0002\u00FE\u0001",  "Nak": "\u0001\u0002\u00FE\u0000"  },  "GroupHeader": "\u0001\u0002\u00FE",  "Feedback": {  "Authorized": "\u0001\u0002\u00FE\u0001",  "Unauthorized": "\u0001\u0002\u00FE\u0000"  }  },  "RequestSoftwareVersionFeedback": {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001\u000F\u00FE"  },  "SystemStatusFeedback": {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001\u000E\u00FE"  },  "AreaInfoFeedback": {  "AckNak": {  "Ack": "\u00FE",  "Nak": "\u00FD"  },  "GroupHeader": "\u0001"  },  "SetResourceStateFeedback":  {  "AckNak":  {  "Ack": "\u00FC",  "Nak": "\u00FD"  },  "GroupHeader": "\u0001"  },  "OutputInfoFeedback": {  "AckNak": {  "Ack": "\u00FE",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "DoorInfoFeedback": {  "AckNak": {  "Ack": "\u00FE",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "DeviceInfoFeedback": {  "AckNak": {  "Ack": "\u00FE",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "RequestAreaResourceStatusFeedback": {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "RequestOutputResourceStatusFeedback": {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "RequestDeviceResourceStatusFeedback": {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "RequestResourceNameFeedback": {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "RequestBypassedResourcesFeedback":  {  "AckNak": {  "Ack": "",  "Nak": ""  },  "GroupHeader": "\u0001"  },  "RequestAlarmFeedback":  {  "AckNak": {  "Ack": "\\u00FE",  "Nak": "\\u00FD"  },  "GroupHeader": "\u0001"  }  } |