Copyright 2026 Synthiam Inc.

EZB Firmware Protocol

Overview

When an I/O controller (Arduino, Microbit, etc.) is programmed with Synthiam firmware compatible with ARC, we refer to it as an EZB. The firmware then allows the EZB I/O controller to use additional capabilities available in ARC. An EZB connects sensors, motors, and peripherals to the computer running ARC.

DIY & Education

Create Custom Firmware

Anyone can use the EZB communication protocol to support a microcontroller or PLC. You can modify existing firmware with new capabilities, or create your own.

EZB Communication Protocol

Enterprise

Become a Hardware Partner

Unleash the full potential of your robot product. Make it compatible with ARC to increase and retain customers. ARC is the industry's proven platform for users to experience the most out of your robot products!

EZB Communication Protocol

Current Supported Hardware

[opt:hardware]

Communication Protocol

Intellectual Property Rights Notice
for Specifications Documentation. 4
Connection. 5
Capability Controller 5
Protocol Commands. 5
Overview.. 5
Release All Servos. 6
Get Unique ID. 6
Set Battery Monitor Protection State. 6
Set Battery Monitor Low Voltage
Threshold. 7
Get Battery Voltage. 7
Get CPU Temperature. 7
UART #0 Init 7
UART #1 Init 7
UART #2 Init 7
UART #0 Write. 8

UART #1 Write. 8
UART #2 Write. 8
UART #0 Get Available Bytes. 8
UART #1 Get Available Bytes. 8
UART #2 Get Available Bytes. 8
UART #0 Get Data from Input Buffer 8
UART #1 Get Data from Input Buffer 9
UART #2 Get Data from Input Buffer 9
I2C Set Clock Speed. 9
Restore to Defaults. 9
I2C Write. 9
I2C Read. 9
Set PWM Duty Cycle. 10
Set Servo Speed. 10
Set Servo Position. 10
Set Digital Port ON.. 10
Set Digital Port OFF. 11
Get Digital Port State. 11
Get ADC Value. 11
Send UART on any Digital Port 12
Streaming Audio. 12
Commands. 12
Sync Example Code. 13
EZBv4Sound Example Code. 13
Camera. 24
Received Packet 24
Sample Code. 24
To-Do. 27

Intellectual Property Rights Notice for Specifications Documentation

.

Technical Documentation. Synthiam Inc. published this specifications document for EZ-B protocols communication and standards support. Additionally, example source-code documents additional inter-protocol relationships and interactions.

.

Copyrights. This documentation is covered by Synthiam Inc. copyrights. Regardless of any other terms that are contained in the terms of use for the Synthiam Inc. website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use Synthiam Inc. technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation.

.

Trade Secrets. The communication protocol published in this document is exclusive to the EZ-B family robot controllers and may not be implemented on third party products without written consent from Synthiam Inc.

.

Patents. Synthiam Inc. may have patents that might cover your implementations of the technologies described in this documentation. Neither this notice nor Synthiam Inc.'s delivery of this documentation grants any licenses under those patents or any other Synthiam Inc. patents. If you would prefer a written license, licenses are available by contacting partners@synthiam.com.

.

Trademarks. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.

.

Fictitious Names. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation may be fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Connection

How to connect to EZ-B. Once the TCP connection has been established, the following character (0x55) **must** be transmitted immediately. One byte will be returned which contains the hardware revision of the EZ-B.

		Notes
Internet Protocol:	TCP	
Default Wi-Fi Mode:	AP Mode (access point)	
Default IP Address:	192.168.1.1	Configurable on EZ-B v4.1/2 and IoTiny
Default Port:	23	Configurable on EZ-B v4.1/2 and IoTiny

Transmit	**Receive**	**Notes**
0x55	EZ-B Hardware Revision (1 byte)	EZB_v4_OS_With_Comm_1 = 4,

		EZB_v4_OS_With_Comm_2 = 42, EZB_v4_OS_IoTiny = 100, Client_Already_Connected = 78, EZB_v3_OS = 166, EZB_v3_BootLoader = 2, **CapabilityController** = 222

Capability Controller

When the EZB responds with the version type, the communication module in ARC will determine what to do next. This is because the version returned will specify the capabilities of the controller to ARC. This allows ARC to know what capabilities the control has, such as HW UART, WiFi ability, Servo ability, etc..
There is an exception with CapaiblityController (222) response. If the EZB responds with CapabilityController, the EZB must be prepared to respond with a Firmware ID. The EZB will send decimal 253 (0xFD) and expect 4 byte UIint32 response.

Transmit	**Receive**	**Notes**
0xFD	Firmware ID (4 bytes) (UInt32)	Firmware IDs and respective cpaabilities are registered with Synthiam. Contact partners@synthiam.com to arrange adding custom supported firmware and capabilities.

The Firmware ID response will register capabilities for the ARC session on the respective EZB connection index. If an ARC program or robot skill attempts to use a capability that is not supported by the Firmware ID, an error will be displayed or logged.

Protocol Commands

Overview

.

All transmit/receive data commands are 8 bit (1 Byte) unless otherwise specified.

.
For additional information regarding supported EZ-B controllers (Arduino, Raspberry Pi, etc) and their firmware, visit https://synthiam.com/GettingStarted/Build-Robots

Release All Servos

Stops PWM and Servo PWM on all digital ports.

Transmit	Receive	Notes
0x01	n/a	

Get Unique ID

Every EZ-B has a unique 12 byte ID that can be queried.

Transmit	Receive	Notes
0x02	12 Bytes	Each byte is an ASCII character of the GUID. GUID is 12 bytes long.

Set Battery Monitor Protection State

Enable or disable the internal EZ-B battery monitor.

Transmit	Receive	Notes
0x04, 0x00, [*0x00* | *0x01*]	n/a	

Set Battery Monitor Low Voltage Threshold

Specify the low voltage threshold for the battery monitor. The value to be specified is a ***little endian 16 bit unsigned INT***, which is the ***VOLTAGE / 0.003862434***.

Transmit	Receive	Notes
0x04, 0x01, *<unsigned 16 bit value>*	n/a	16 bit value is little endian.

Get Battery Voltage

Returns the battery voltage of the EZ-B. The returned value is a ***little endian unsigned 16 bit INT***. To get voltage, multiple the value by 0.003862434m. This means the ***voltage = RETURN VALUE * 0.003862434***.

Transmit	Receive	Notes

0x04, 0x02	2 bytes <*unsigned 16 bit value*>	16 bit value is little endian.

Get CPU Temperature

Returns the temperature of the EZ-B CPU. The returned value is a ***little endian unsigned 16 bit INT***. To get temperature in Celsius, multiple the value by 0.026341480261472. This means the ***temperature = RETURN VALUE * 0.026341480261472***.

Transmit	Receive	Notes
0x04, 0x03	2 bytes <*unsigned 16 bit value*>	16 bit value is little endian.

UART #0 Init

Initialize the UART #0 port on the EZ-B with the ***specified baud rate*** as a ***little endian unsigned 32 bit INT***. This also clears the UART input buffer on the EZ-B.

Transmit	Receive	Notes
0x04, 0x05, <*unsigned 32 bit value*>	n/a	32 bit baud rate value is little endian.

UART #1 Init

Initialize the UART #1 port on the EZ-B with the ***specified baud rate*** as a ***little endian unsigned 32 bit INT***. This also clears the UART input buffer on the EZ-B.

Transmit	Receive	Notes
0x04, 0x09, <*unsigned 32 bit value*>	n/a	32 bit baud rate value is little endian.

UART #2 Init

Initialize the UART #2 port on the EZ-B with the ***specified baud rate*** as a ***little endian unsigned 32 bit INT***. This also clears the UART input buffer on the EZ-B.

Transmit	Receive	Notes
0x04, 0x0D, <*unsigned 32 bit value*>	n/a	32 bit baud rate value is little endian.

UART #0 Write

Write data out of the UART #0 port on the EZ-B. The length of the data is specified as a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x06, *<unsigned 16 bit value data length>*, *DATA BYTES…*	n/a	16 bit value is little endian.

UART #1 Write

Write data out of the UART #1 port on the EZ-B. The length of the data is specified as a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x0A, *<unsigned 16 bit value data length>*, *DATA BYTES…*	n/a	16 bit value is little endian.

UART #2 Write

Write data out of the UART #2 port on the EZ-B. The length of the data is specified as a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x0E, *<unsigned 16 bit value data length>*, *DATA BYTES…*	n/a	16 bit value is little endian.

UART #0 Get Available Bytes

Get the available number of bytes in the UART #0 input buffer. The returned length is a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x07	2 Bytes <unsigned 16 bit>	16 bit returned value is little endian.

UART #1 Get Available Bytes

Get the available number of bytes in the UART #1 input buffer. The returned length is a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x0B	2 Bytes <unsigned 16 bit>	16 bit returned value is little endian.

UART #2 Get Available Bytes

Get the available number of bytes in the UART #2 input buffer. The returned length is a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x0F	2 Bytes <unsigned 16 bit>	16 bit returned value is little endian.

UART #0 Get Data from Input Buffer

Get the specified number of bytes from the UART #0 input buffer. The specified length is a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x08, *<unsigned 16 bit length to receive>*	*<Multiple bytes as specified by the transmit call>*	16 bit value is little endian.

UART #1 Get Data from Input Buffer

Get the specified number of bytes from the UART #1 input buffer. The specified length is a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x0C, *<unsigned 16 bit length to receive>*	*<Multiple bytes as specified by the transmit call>*	16 bit value is little endian.

UART #2 Get Data from Input Buffer

Get the specified number of bytes from the UART #2 input buffer. The specified length is a ***little endian unsigned 16 bit INT***.

Transmit	Receive	Notes
0x04, 0x10, *<unsigned 16 bit length to receive>*	*<Multiple bytes as specified by the transmit call>*	16 bit value is little endian.

I2C Set Clock Speed

Specify the clock speed of the i2c interface. The specified length is a ***little endian unsigned 32 bit INT***.

Transmit	Receive	Notes
0x04, 0x11, *<unsigned 32 bit speed>*	*n/a*	32 bit value is little endian.

Restore to Defaults

Restore the EZ-B to default factory settings.

Transmit	Receive	Notes

0x04, 0x13	*n/a*	32 bit value is little endian.

I2C Write

Write data out of the I2C port on the EZ-B. The length of the data is specified as an ***unsigned 8 bit INT***.

Transmit	Receive	Notes
0x0A, <unsigned 8 bit byte i2c address>, *<unsigned 8 bit bytes to send>*, *DATA BYTES…*	n/a	

I2C Read

Get the specified number of bytes from the I2C input buffer. The specified length is an ***unsigned 8 bit INT***.

Transmit	Receive	Notes
0x0B, <unsigned 8 bit byte i2c Address>, *<unsigned 8 bit byes to read>*	*<Multiple bytes as specified by request>*	

Set PWM Duty Cycle

Specify the PWM Duty Value to be outputted on the specified digital port. The *Port Index* is the digital port to be outputted starting with D0. The PWM value is between 0 and 100.
Example of 0% duty on D3: *0x12, 0x00*
Example of 75% duty on D8: *0x17, 0x4B*

Transmit	Receive	Notes
0x0F + *<Port Index>*, *<unsigned 8 bit duty>*	*n/a*	

Set Servo Speed

Specify the Servo Speed for the specified digital port. The *Port Index* is the digital port to be outputted starting with D0.
*Note: A servo position must be specified before the servo speed can be set.
Example of setting servo speed to 3 on D3: *0x2A, 0x03*

Transmit	Receive	Notes
0x27 + *<Port Index>*, *<unsigned 8 bit speed>*	*n/a*	

Set Servo Position

Specify the Servo Position for the specified digital port.
The *Port Index* is the digital port to be outputted starting with D0. The position is in degrees between 1 and 180 (specify 0 to release the servo PWM).
Example of setting servo on Port D3 to position 90: *0xAF, 0x5A*
Example of setting servo on Port D18 to position 145: *0xBE, 0x91*

Transmit	Receive	Notes
0xAC + *<Port Index>*, *<unsigned 8 bit position>*	*n/a*	

Set Digital Port ON

Specify the Digital Port to be in the ON (true) state on the specified digital port. The *Port Index* is the digital port to be outputted starting with D0.
Example of TRUE on D3: *0x67*
Example of TRUE on D8: *0x6C*

Transmit	Receive	Notes
0x64 + *<Port Index>*	*n/a*	

Set Digital Port OFF

Specify the Digital Port to be in the OFF (false) state on the specified digital port. The *Port Index* is the digital port to be outputted starting with D0.
Example of TRUE on D3: *0x7F*
Example of TRUE on D8: *0x84*

Transmit	Receive	Notes
0x7C + *<Port Index>*	*n/a*	

Get Digital Port State

Get the state of the specified digital port. The *Port Index* is the digital port to starting with D0. Once the port is read, this puts the port into INPUT mode.

Example of getting value of port D3: *0x97*
Example of TRUE on D8: *0x9C*

Transmit	Receive	Notes
0x94 + *<Port Index>*	*[0x00 | 0x01]*	Returns a 0 (false) or 1 (true) of the state of the digital port.

Get ADC Value

Get the ADC Value of the specified analog port. The *Port Index* is the ADC (analog) port to be read starting with ADC0.
Example of reading ADC value on port ADC3: 0xC7

Transmit	Receive	Notes
0xC4 + *<Port Index>*	*<unsigned 16 bit INT>*	

Send UART on any Digital Port

Send the specified data to any digital port at one of the supported baud rates.

Baud_4800 = 0,
Baud_9600 = 1,
Baud_19200 = 2,
Baud_38400 = 3,
Baud_57600 = 4,
Baud_115200 = 5

Example of writing '*HELLO*' to port D3 at 9600:
0xCF, 0x01, 0x05, 0x48, 0x45, 0x4C, 0x4C, 0x4F

Transmit	Receive
0xCC + *<Port Index>, <unsigned 8 bit value baud rate>, <unsigned 8 bit value data size>, DATA…*	*n/a*

Streaming Audio

The EZ-B uses a raw 8 bit mono PCM stream at 14.7khz (14,700hz) for the DAC to play audio. The EZ-B has an internal 50k buffer for audio. The audio data is sent to the EZ-B and the DAC can be enabled to begin playing the data within the buffer. Once the DAC is enabled, you must continue streaming data at the EZ-B at the rate of 14.7khz. This can be done using a calculation to determine how long to pause for.

Commands

Load data into the EZ-B buffer

```
// build
  the packet including the LOAD command (0x01) and audio data bytes
List<byte> dataTmp = new List<byte>();
dataTmp.Add(0x01);
dataTmp.AddRange(BitConverter.GetBytes((UInt16)audioData));
dataTmp.AddRange(bTmp.Take(audioData));

// send
  the constructed packet to the ez-b
ezb.sendCommand(0xfe,
  dataTmp.ToArray());
```

Begin playing the EZ-B's audio buffer

```
//
  instruct the ezb to begin playing the DAC audio buffer
ezb.sendCommand(0xfe,
  0x02);
```

Stop the EZ-B's DAC audio play

```
ezb.sendCommand(0xfe,
  0x00);
```

Sync Example Code

To sync your client with the playback frequency of the EZ-B (14.7khz), this is an example code.

```
// Use
  the stop watch to determine how long we have been playing for
// the
  stop watch begins at the start of the playback (after initial load of data)
float runtime = (float)sw.ElapsedMilliseconds;

//
  convert milliseconds to seconds. Multiple the seconds by the audio bit rate.
// this
  gets us where we should be in the stream.
// We
  add the prebuffer length because we don't actually wait and play during the prebuffer,
  as the pre buffer is transmitted in one chunk with no delays.
// Also,
  the stop watch doesn't start until the audio begins to play after prebuffer.
float
  shouldBeAtSamplePosition = ((runtime / 1000f) * (float)AUDIO_SAMPLE_BITRATE)
  + (float)_afterBuffer.Length;

// check
  to see if we are behind on the stream based on how long the audio has been
  playing in samples.
float difference =
```

```
  _position - shouldBeAtSamplePosition;

// if
  there is a difference, wait some time and let the ez-b buffer play and catch
  up
if (difference > 0)
  {

  //
  convert the sample difference to a millisecond time so we know how long to
  sleep and let the ez-b catch up
  float delay = (difference
  / AUDIO_SAMPLE_BITRATE) * 1000;


  Thread.Sleep((int)delay);
}
```

EZBv4Sound Example Code

This is a C# example of how to play back an audio stream to
the EZ-B for your reference

```
using System;
using
  System.Collections.Generic;
using
  System.Diagnostics;
using
  System.IO;
using
  System.Linq;
using
  System.Threading;

namespace
  EZ_B {

  public
  class EZBv4Sound : DisposableBase {

    class
  ThreadStartParams {

      public
  int CNT;
      public
  MemoryStream AudioStream;

      public
  ThreadStartParams(int cnt, MemoryStream audioStream) {

        CNT =
```

```
cnt;

AudioStream = audioStream;
   }
  }


EZB            _ezb;

EZTaskScheduler  _tsSound        = null;

int            _playFromSample  = 0;
  volatile
int     _cnt           = 0;

int            _position       = 0;
  bool
_isPlaying       = false;

  byte []
_rampUp   = new byte[256];
  byte []
_rampDown = new byte[256];

  //
Appended to end of all audio streams to compensate for the time it takes to
pre-buffer
  byte
[]         _afterBuffer;

  ///
<summary>
  /// The
recommended size of the the audio packets
  ///
</summary>
  public
static readonly int RECOMMENDED_PACKET_SIZE = 256;

  ///
<summary>
  /// The
recommended size of the prebuffer before playing the audio
  ///
</summary>
  public
static readonly int RECOMMENDED_PREBUFFER_SIZE = 20000;

  ///
<summary>
  /// The
sample rate at which the data is played back on the EZ-B
  ///
</summary>
```

```
  public
static readonly int AUDIO_SAMPLE_BITRATE = 14700;

  ///
<summary>
  /// The
size of each packet which is transmitted over the wire to the EZ-B.
  ///
</summary>
  public
int PACKET_SIZE = RECOMMENDED_PACKET_SIZE;

  ///
<summary>
  /// The
ammount of data to prebuffer to the EZ-B before playing the audio. The EZ-B
has a 50k buffer, so this value cannot be any higher than that.
  ///
</summary>
  public
int PREBUFFER_SIZE = RECOMMENDED_PREBUFFER_SIZE;

  ///
<summary>
  /// Event
exceuted when new data is being sent to the EZ-B
  ///
</summary>
  public
delegate void OnAudioDataHandler(int minVal, int maxVal, int avgVal);

  ///
<summary>
  /// Event
executed when the volume value has changed
  ///
</summary>
  public
delegate void OnVolumeChangedHandler(decimal volume);

  ///
<summary>
  /// Event
executed when the audio has stopped playing
  ///
</summary>
  public
delegate void OnStopPlayingHandler();

  ///
<summary>
  /// Event
executed when the audio has begun playing
  ///
```

```
</summary>
  public
delegate void OnStartPlayingHandler();

  ///
<summary>
  /// Event
executed when the audio level is clipping. This means the volume value is
amplifying the audio past the limits
  ///
</summary>
  public
delegate void OnClippingStatusHandler(bool isClipping);

  public
event OnAudioDataHandler OnAudioDataChanged;
  public
event OnVolumeChangedHandler OnVolumeChanged;
  public
event OnStopPlayingHandler OnStopPlaying;
  public
event OnStartPlayingHandler OnStartPlaying;
  public
event OnClippingStatusHandler OnClippingStatus;

  private
decimal _volume             = 100;

  ///
<summary>
  ///
Returns status if music is playing
  ///
</summary>
  public
bool IsPlaying {
    get {

return _isPlaying;
    }
  }

  ///
<summary>
  /// Get
or Set the volume
  ///
</summary>
  public
decimal Volume {
    get {

return _volume;
    }
```

```
    set {

      if
(_volume != value && OnVolumeChanged != null)

OnVolumeChanged(value);


_volume = value;
    }
  }

  protected
internal EZBv4Sound(EZB ezb) {

    _ezb =
ezb;

    for
(int x = 0; x < _rampUp.Length; x++)
      _rampUp[x]
= (byte)(x / 2);

    for
(int x = 0; x < 64; x++)

_rampDown[x] = (byte)(128 - (x * 2));


initAfterBufferArrays();


_tsSound = new EZTaskScheduler("v4 Sound");

_tsSound.OnEventToRun += threadStreamAudio;

_tsSound.OnEventStart += _tsSound_OnEventStart;

_tsSound.OnEventCompleted += _tsSound_OnEventCompleted;
  }

  private
void _tsSound_OnEventStart(int taskId, object o) {


_isPlaying = true;
  }

  private
void _tsSound_OnEventCompleted(int taskId, object o) {


_isPlaying = false;
  }
```

```
  void
initAfterBufferArrays() {

    if
(_afterBuffer == null || _afterBuffer.Length != PREBUFFER_SIZE) {


_afterBuffer = new byte[PREBUFFER_SIZE];

      for
(int x = 0; x < _afterBuffer.Length; x++)

_afterBuffer[x] = 0;
    }
  }

  ///
<summary>
  /// Play
the Audio Data out of the EZ-B.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
</summary>
  public
void PlayDataWait(byte[] data) {


PlayDataWait(null, data, _volume, new int[] { }, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
</summary>
  public
void PlayData(byte[] data) {


PlayData(null, data, _volume, new int[] { }, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
```

```
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
</summary>
  public
void PlayData(byte[] data, decimal volume) {


PlayData(null, data, volume, new int[] { }, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
</summary>
  public
void PlayData(byte[] data, decimal volume) {


PlayData(null, data, volume, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
</summary>
  public
void PlayDataWait(byte[] data, decimal volume) {


PlayDataWait(null, data, volume, new int[] { }, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
```

```
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayData(Stream ms) {


PlayData(ms, null, _volume, new int[] { }, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayData(Stream ms) {


PlayData(ms, null, _volume, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public void
PlayData(Stream ms, int playFromSample) {


PlayData(ms, null, _volume, playFromSample);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
```

```
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayDataWait(Stream ms) {


PlayDataWait(ms, null, _volume, new int[] { }, 0);
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayData(Stream ms, decimal volume) {


PlayData(ms, null, volume, new int[] { }, 0);
  }

  /// <summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayData(Stream ms, decimal volume) {


PlayData(ms, null, volume, 0);
  }

  ///
<summary>
  ///
```

```
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayData(Stream ms, byte[] bytes, decimal volume, int playFromSample) {

    Stop();

    if
(!_ezb.IsConnected)

return;

    if
(_ezb.EZBType != EZB.EZ_B_Type_Enum.ezb4) {


_ezb.Log(false, "This feature is only available for EZ-B v4");


return;
    }


initAfterBufferArrays();


MemoryStream tmpMs = new MemoryStream();

    if
(_ezb.GetFirmwareVersionEnum() == EZB.FirmwareVersionEnum.EZB_v4_OS_IoTiny)

tmpMs.Write(_rampUp, 0, _rampUp.Length);

    if (ms
!= null)

ms.CopyTo(tmpMs);
    else if
(bytes != null)

tmpMs.Write(bytes, 0, bytes.Length);
    else
      throw
new Exception("Expecting either a memorystream or byte array of
audio");

    if
```

```
(_ezb.GetFirmwareVersionEnum() == EZB.FirmwareVersionEnum.EZB_v4_OS_IoTiny)

tmpMs.Write(_rampDown, 0, _rampDown.Length);

    if
(tmpMs.Length == 0)

return;


tmpMs.Write(_afterBuffer, 0, _afterBuffer.Length);


_playFromSample = playFromSample;


tmpMs.Position = playFromSample;

    Volume
= volume;

    _cnt++;


_tsSound.StartNew(new ThreadStartParams(_cnt, tmpMs));
  }

  ///
<summary>
  ///
Stream raw audio data to the EZ-B v4's speakers.
  /// 0 is
silent, 100 is normal, 200 is 2x gain, 300 is 3x gain, etc.
  /// The
audio must be RAW 8 Bit at 18 KHZ Sample Rate
  ///
*Note: You must dispose of the memory stream yourself after calling this
  ///
</summary>
  public
void PlayDataWait(Stream ms, byte[] bytes, decimal volume, int
playFromSample) {

    Stop();

    if
(!_ezb.IsConnected)

return;

    if
(_ezb.EZBType != EZB.EZ_B_Type_Enum.ezb4) {
```

```
_ezb.Log(false, "This feature is only available for EZ-B v4");


return;
    }


initAfterBufferArrays();


MemoryStream tmpMs = new MemoryStream();

    if
(_ezb.GetFirmwareVersionEnum() == EZB.FirmwareVersionEnum.EZB_v4_OS_IoTiny)
      tmpMs.Write(_rampUp,
0, _rampUp.Length);

    if (ms
!= null)

ms.CopyTo(tmpMs);
    else if
(bytes != null)

tmpMs.Write(bytes, 0, bytes.Length);
    else
      throw
new Exception("Expecting either a memorystream or byte array of
audio");

    if
(_ezb.GetFirmwareVersionEnum() == EZB.FirmwareVersionEnum.EZB_v4_OS_IoTiny)

tmpMs.Write(_rampDown, 0, _rampDown.Length);

    if
(tmpMs.Length == 0)

return;


tmpMs.Write(_afterBuffer, 0, _afterBuffer.Length);


_playFromSample = playFromSample;


tmpMs.Position = playFromSample;

    Volume
= volume;

    _cnt++;
```

```
threadStreamAudio(0, new ThreadStartParams(_cnt, tmpMs));
  }

  void
threadStreamAudio(int taskId, object o) {


ThreadStartParams threadStartParams = (ThreadStartParams)o;

    if
(_cnt != threadStartParams.CNT || !_ezb.IsConnected)

return;


Stopwatch sw = new Stopwatch();

    try {

      if
(OnStartPlaying != null)

OnStartPlaying();


_ezb.sendCommand(EZB.CommandEnum.CmdSoundStreamCmd,
(byte)EZB.CmdSoundv4Enum.CmdSoundInitStop);

      bool
playing = false;

      _position
= 0;


byte[] bTmp = new byte[PREBUFFER_SIZE];

      do {

        int
bytesRead;

        if
(playing)

bytesRead = threadStartParams.AudioStream.Read(bTmp, 0, PACKET_SIZE);

else

bytesRead = threadStartParams.AudioStream.Read(bTmp, 0, PREBUFFER_SIZE);


_position += bytesRead;
```

```
bool isClipping = false;
      int
highest = 0;
      int
lowest = 255;
      int
average = 0;

long total = 0;

decimal volumeMultiplier = _volume / 100m;

      if
(_volume != 100 || OnAudioDataChanged != null)

for (int x = 0; x < bytesRead; x++) {


decimal newVal = (decimal)bTmp[x];


if (newVal > 128)

newVal = Math.Max(128, 128 + ((newVal - 128) * volumeMultiplier));

else if (newVal < 128)

newVal = Math.Min(128, 128 - ((128 - newVal) * volumeMultiplier));


if (newVal > 255) {


newVal = 255;


isClipping = true;

} else if (newVal < 0) {


newVal = 0;


isClipping = true;

}


highest = Math.Max(highest, (int)newVal);

lowest = Math.Min(lowest, (int)newVal);
```

```
total += (int)newVal;


bTmp[x] = (byte)newVal;
        }


average = (int)(total / bytesRead);

       List<byte>
dataTmp = new List<byte>();

dataTmp.Add((byte)EZB.CmdSoundv4Enum.CmdSoundLoad);

dataTmp.AddRange(BitConverter.GetBytes((UInt16)bytesRead));

dataTmp.AddRange(bTmp.Take(bytesRead));

       if
(_cnt != threadStartParams.CNT)

return;


_ezb.sendCommand(EZB.CommandEnum.CmdSoundStreamCmd, dataTmp.ToArray());

       if
(_cnt != threadStartParams.CNT)

return;

       if
(!playing && _position >= PREBUFFER_SIZE) {


_ezb.sendCommand(EZB.CommandEnum.CmdSoundStreamCmd,
(byte)EZB.CmdSoundv4Enum.CmdSoundPlay);


playing = true;


sw.Start();
       }

       if
(OnAudioDataChanged != null)

OnAudioDataChanged(lowest, highest, average);

       if
(playing) {
```

```
if (OnClippingStatus != null)

OnClippingStatus(isClipping);


float runtime = (float)sw.ElapsedMilliseconds;


// convert milliseconds to seconds. Multiple the seconds by the audio bit
rate.

// this gets us where we should be in the stream.

// We add the prebuffer lengthh because we don't actually wait and play
during the prebuffer, as the pre buffer is transmitted in one chunk with no
delays.

// Also, the stop watch doesn't start until the audio begins to play after
prebuffer.

float shouldBeAtSamplePosition = ((runtime / 1000f) *
(float)AUDIO_SAMPLE_BITRATE) + (float)_afterBuffer.Length;


// check to see if we are behind on the stream based on how long the audio
has been playing in samples.

float difference = _position - shouldBeAtSamplePosition;


// if there is a difference, wait some time and let the ez-b buffer play and
catch up

if (difference > 0) {


// convert the sample difference to a millisecond time so we know how long to
sleep and let the ez-b catch up

float delay = (difference / AUDIO_SAMPLE_BITRATE) * 1000;


Thread.Sleep((int)delay);
        }
      }
    }
while (_position < threadStartParams.AudioStream.Length &&
_ezb.IsConnected && _cnt == threadStartParams.CNT);

   } catch
(Exception ex) {
```

```
_ezb.Log(false, "Error Streaming Audio: {0}", ex);
    }
finally {


sw.Stop();


threadStartParams.AudioStream.Dispose();

      if
(_ezb.IsConnected)

_ezb.sendCommand(EZB.CommandEnum.CmdSoundStreamCmd,
(byte)EZB.CmdSoundv4Enum.CmdSoundInitStop);

      if
(OnClippingStatus != null)

OnClippingStatus(false);

      if
(OnStopPlaying != null)

OnStopPlaying();
    }
  }

  ///
<summary>
  /// Stop
the audio which is being played
  ///
</summary>
  public
void Stop() {

    _cnt++;

    // a
slight amount of time to ensure the audio has stopped before attempting to
play new audio

System.Threading.Thread.Sleep(250);
  }

  ///
<summary>
  ///
Dispose of the AutoPositioner
  ///
</summary>
  protected
override void DisposeOverride() {
```

```
  OnStartPlaying = null;

  OnStopPlaying = null;


  _tsSound.Dispose();

  _tsSound = null;

     _cnt++;
   }
  }
}
```

Camera

The camera protocol is a stream of JPG images, each with a header.

Received Packet

Visible image (regular camera): EZIMG<unsigned 16bit int length><jpg data>
Infrared image (IPS camera): EZIRC<unsigned 16bit int length><jpg data>

Sample Code

This is example code from the EZ-B SDK that loops for ever collecting video data, unless there is a disconnect or timeout. In your program, you will want to handle connect and disconnects gracefully. This code is presented as an example only.

```
   public class EZBv4VideoSample {

     /// <summary>
     /// Event raised when
  an infrared image is ready. This image must be disposed after use.
     /// </summary>
     public event
  OnImageIRReadyHandler OnImageIRReady;
     public delegate void OnImageIRReadyHandler(Bitmap BM);

     /// <summary>
     /// Event raised when
  the image is ready. This image must be disposed after use.
     /// </summary>
     public event OnImageReadyHandler
  OnImageReady;
     public delegate void OnImageReadyHandler(Bitmap BM);

     /// <summary>
     /// Event raised when
```

```
 an infrared image is ready.
    /// </summary>
    public event
 OnImageIRDataReadyHandler OnImageIRDataReady;
    public delegate void OnImageIRDataReadyHandler(byte[] imageData);

    /// <summary>
    /// Event raised when
 the image is ready.
    /// </summary>
    public event
 OnImageDataReadyHandler OnImageDataReady;
    public delegate void OnImageDataReadyHandler(byte[] imageData);

    readonly byte [] TAG_EZIMAGE = new byte[] { (byte)'E', (byte)'Z', (byte)'I', (byte)'M',
(byte)'G' };
    readonly byte [] TAG_EZIRC = new byte[] { (byte)'E', (byte)'Z', (byte)'I', (byte)'R',
(byte)'C' };

    readonly int BUFFER_SIZE =
 128000;

    void imageThreadWorker()
 {


 List<byte> bufferImage = new List<byte>();
     byte[] bufferTmp = new byte[BUFFER_SIZE];

     try {


 using (var tcpClient = new TcpClient()) {


 IAsyncResult ar = tcpClient.BeginConnect(threadStartParams.IPAddress,
 threadStartParams.Port, null, null);


 if
 (!ar.AsyncWaitHandle.WaitOne(TimeSpan.FromSeconds(3), false))

 throw new TimeoutException();


 tcpClient.EndConnect(ar);


 tcpClient.ReceiveBufferSize = BUFFER_SIZE;

 tcpClient.NoDelay = true;

 tcpClient.ReceiveTimeout = 5000;
```

```
tcpClient.SendTimeout = 3000;


using (NetworkStream ns =
tcpClient.GetStream())

while
(tcpClient.Connected) {


int read =
ns.Read(bufferTmp, 0, BUFFER_SIZE);


if (read == 0)

throw new Exception("Client
disconnected");


bufferImage.AddRange(bufferTmp.Take(read));


if (bufferImage.Count
> 512000)

throw new Exception(string.Format("Image data is
piling up and not being processed. We stopped the collection at {0:###,###}
Bytes. Post on the forum so we can better understand what is happening and
fix it.", bufferImage.Count));


LOOP_AGAIN:

ImageTypeEnum imageType = ImageTypeEnum.NA;

int foundStart = -1;


if (bufferImage.Count
< TAG_EZIMAGE.Length)

continue;


for (int p = 0; p <
bufferImage.Count - TAG_EZIMAGE.Length; p++)

if (bufferImage[p] ==
TAG_EZIMAGE[0] &&

bufferImage[p + 1] == TAG_EZIMAGE[1] &&

bufferImage[p + 2] == TAG_EZIMAGE[2] &&
```

```
bufferImage[p + 3] == TAG_EZIMAGE[3] &&

bufferImage[p + 4] == TAG_EZIMAGE[4]) {


imageType = ImageTypeEnum.Camera;


foundStart = p;


break;

} else if (bufferImage[p] ==
TAG_EZIRC[0] &&

bufferImage[p + 1] == TAG_EZIRC[1] &&

bufferImage[p + 2] == TAG_EZIRC[2] &&

bufferImage[p + 3] == TAG_EZIRC[3] &&

bufferImage[p + 4] == TAG_EZIRC[4]) {


imageType = ImageTypeEnum.Infrared;


foundStart = p;


break;

}


if (foundStart == -1)

continue;


if (foundStart > 0)

bufferImage.RemoveRange(0, foundStart);


if (bufferImage.Count
< TAG_EZIMAGE.Length + sizeof(UInt32))

continue;


int imageSize =
```

```
(int)BitConverter.ToUInt32(bufferImage.GetRange(TAG_EZIMAGE.Length,
  sizeof(UInt32)).ToArray(),
  0);


  if (bufferImage.Count
  <= imageSize + TAG_EZIMAGE.Length + sizeof(UInt32))

  continue;


  bufferImage.RemoveRange(0, TAG_EZIMAGE.Length + sizeof(UInt32));


  _imageSize = imageSize;


  try {


  if (imageType ==
  ImageTypeEnum.Camera) {


  if (OnImageReady != null)

  OnImageReady(new Bitmap(new MemoryStream(bufferImage.GetRange(0,
imageSize).ToArray())));


  if (OnImageDataReady
  != null)

  OnImageDataReady(bufferImage.GetRange(0, imageSize).ToArray());

  } else if (imageType ==
  ImageTypeEnum.Infrared) {


  if (OnImageIRReady != null)

  OnImageIRReady(new Bitmap(new MemoryStream(bufferImage.GetRange(0,
imageSize).ToArray())));


  if (OnImageIRDataReady
  != null)

  OnImageIRDataReady(bufferImage.GetRange(0, imageSize).ToArray());

  }

  } catch (Exception ex) {
```

```
_ezb.Log(false, "ezbv4 camera
image render error: {0}", ex);

}


bufferImage.RemoveRange(0, imageSize);


// If
there's at least 5kb of data in the buffer, loop again and see if there's
another image in the buffer

//
Without doing this, the there's a chance the buffer will fill with future
images and we'll never catch up because image data is only processed when an
image is available

if (bufferImage.Count
> 5000)

goto LOOP_AGAIN;

}

}

} catch (Exception ex) {


_ezb.Log(false, "EZ-B v4
Camera Error: {0}", ex);

}
   }
  }
```

To-Do

EZ-B protocol commands not yet specified in this document include…

-

Ultrasonic Distance Sensor

-

Built-in Audio Samples

Extend EZB Protocol

Expanding the capabilities of your robotic projects with Synthiam's ARC platform is made possible through the innovative use of the EZB Protocol Command 0, specifically designed to

extend EZBs with new custom functionality. This unique command is a foundational gateway for introducing custom functionalities into the EZB firmware, empowering users to transcend beyond the standard suite of commands. By leveraging this protocol, enthusiasts and developers alike can craft bespoke commands tailored to their specific needs, thus unlocking a new realm of possibilities. Through detailed examples and comprehensive guides, from DJ's UNO Firmware enhancements to the advanced Balance Robot project, the pathway is clear for those aspiring to create or enhance robot skills. This initiative fosters creativity and broadens the scope of what can be achieved, ensuring that your robotic endeavors are limited only by imagination.

To add new functions to the EZB firmware for the Arduino Due/Mega, you can use the EZB Protocol Command 0, also documented in some firmware examples as `CmdOurCustomCmds` (Command 0). This is a starting point for creating custom commands. The EZB Command 0 is unused for the standard built-in commands of ARC (i.e., moving servos, reading/writing digital I/O, etc.), so it is a designated area in the firmware for user-defined commands, which helps avoid conflicts with existing or future ARC commands.

Several examples of extending the standard EZB Communication Protocol with custom commands exist. Please use these examples as a reference when building your custom EZB firmware...

- Robot Skill Tutorial for extending protocol has example code and several additional details.
- DJ's UNO Firmware with Wheel Encoder Counter has a corresponding robot skill Wheel Encoder Counter. A video documenting the entire process of creating the robot skill and firmware is here.
- The Balance Robot firmware for the Arduino Mega is here: https://synthiam.com/Support/Hardware/Balance-Robot. This example extends the EZB Protocol Command 0 with new commands.
  It uses the Inverted Pendulum Movement Panel robot skill.

Creating a custom robot skill will most likely be necessary to send the commands to your new firmware EZB device. You can use the Create Robot Skill Tutorial to learn how to create a robot skill.

This live hack video by DJ Sures has plenty of good information as he creates a robot skill and extends the EZB protocol with new commands. This example demonstrates how to write and read from the protocol with custom commands. It is a long video, but you can skip through sections to find relevant lessons.

Here's a step-by-step guide on how to define new custom command functions in the Arduino Mega EZB firmware:

1.

Identify the Custom Command Range:

Ensure that the custom command numbers you choose do not conflict with existing commands in the firmware. Defining a range of numbers for your custom commands unique to your project is a good practice.
Adding your custom commands under the EZB Protocol Command 0 will prevent conflict with existing EZB commands documented in the protocol datasheet.

2.
Modify the Firmware:

Open the Arduino Mega EZB firmware source code in the Arduino IDE or your preferred development environment.
Locate the section of the code that handles the `CmdOurCustomCmds` command.

Define your custom command numbers as constants or enums for better code readability. For example:

```
const byte CmdCustomAction1 = 1;
const byte CmdCustomAction2 = 2;
// ... and so on
```

Add additional `else if` statements within the `CmdOurCustomCmds` block to handle your new custom commands. For example:

```
} else if (cmd == CmdOurCustomCmds) {
  byte cmd2 = ReadByte();
  if (cmd2 == CmdCustomAction1) {
    // Handle custom action 1
  } else if (cmd2 == CmdCustomAction2) {
    // Handle custom action 2
  }
  // ... additional custom commands
}
```

Implement the functions that correspond to each custom command. These functions will contain the logic for the actions you want to perform when the command is received.

3.
Compile and Upload the Firmware:

After adding your custom commands and their corresponding functions, compile the firmware.
Upload the compiled firmware to your Arduino Mega.

4. **Update Your ARC Skill:**

In your ARC skill, update the code to send the custom commands to the Arduino

Mega. Use the `CmdOurCustomCmds` command followed by the specific custom command number you defined.

For example, to send `CmdCustomAction1`, you would send a packet with `CmdOurCustomCmds` followed by `CmdCustomAction1`.

5. Testing:

Test your ARC skill to ensure that the custom commands are being sent correctly and that the Arduino Mega is responding as expected.

Regarding the number range for custom commands, there isn't a specific range reserved for user-defined commands. However, choosing numbers that are not already in use by the firmware and are unlikely to be used in future updates is essential. Using sub-commands under `CmdOurCustomCmds` is a good approach, as it organizes your custom commands under a single parent command and reduces the likelihood of conflicts.

Remember to document your custom commands and maintain a list of them to ensure that you don't accidentally reuse numbers for different functions in the future.