Contents

[EV3 Arduino Extensions 2](#_Toc433695665)

[Create a new project. 2](#_Toc433695666)

[Playing Tracks 6](#_Toc433695667)

[Led Strips 7](#_Toc433695668)

[Limitations 8](#_Toc433695669)

[Block Reference 8](#_Toc433695670)

[General Blocks 8](#_Toc433695671)

[ExReset 8](#_Toc433695672)

[Music Blocks 9](#_Toc433695673)

[ExSetMusicMode 9](#_Toc433695674)

[ExPlayTrack 9](#_Toc433695675)

[ExIsTrackComplete 10](#_Toc433695676)

[ExStopTrack 10](#_Toc433695677)

[MIDI 10](#_Toc433695678)

[Led Blocks 11](#_Toc433695679)

[ExConfigureLeds 11](#_Toc433695680)

[ExShowLeds 11](#_Toc433695681)

[ExSetLed 11](#_Toc433695682)

[ExSetLedRangeFade 12](#_Toc433695683)

[ExSetLedBarGraph 12](#_Toc433695684)

[ExSetLedRangeArray 12](#_Toc433695685)

[ExRotateAllLeds 13](#_Toc433695686)

[ExWalkAllLeds 13](#_Toc433695687)

[ExWalkLedFrames 13](#_Toc433695688)

[Low Level Blocks 13](#_Toc433695689)

[SendCmdAndWait 14](#_Toc433695690)

[IsStatusClear 14](#_Toc433695691)

[Compiling the Ardiuno 14](#_Toc433695692)

[Getting Ready 14](#_Toc433695693)

[Building 15](#_Toc433695694)

[Memory Conservation 15](#_Toc433695695)

[Features 15](#_Toc433695696)

[Verbosity 16](#_Toc433695697)

[SD Library 17](#_Toc433695698)

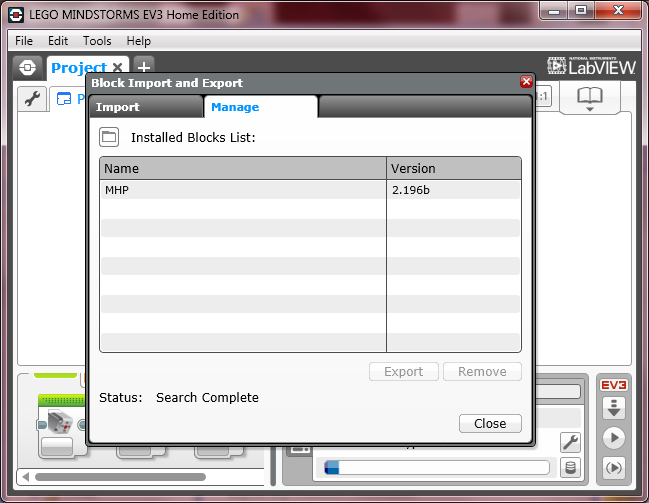
[EV3 <-> Arduino I2C Protocol 17](#_Toc433695699)

# EV3 Arduino Extensions

## Create a new project.

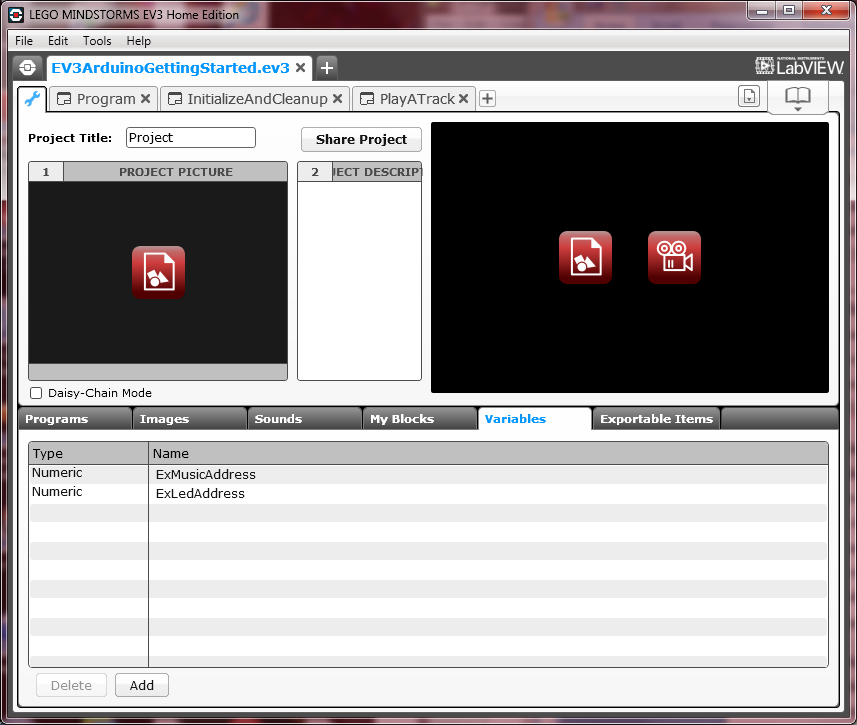
Create a new project by going to the menu in the upper left hand corner and selecting ‘File’, then ‘New Project’. If you already have a project you are working on that you would like to add use of the Arduino Extensions to, you can open that.

First we need to import some blocks that allow us to communicate with the Arduino board. These blocks will show up in the Yellow group with the sensors. Go back to the menu in the upper left and select ‘Tools’ and then ‘Block Import’. When the dialog box comes up, select the ‘Manage’ tab. It should list ‘MHP’ and version ‘2.196b’ like this:



If it does, then you can press ‘Close’ and skip the rest of this paragraph. If it does not, select the ‘Import’ tab, press the ‘Browse’ button, navigate to the folder that contains the distribution files from Mr. Miley, select the file MHP2196b.ev3b, and press ‘Open’. Back at the dialog box, press ‘Import’. Finally, press ‘Close’. This only needs to be done once on a given laptop.

There are a couple of variables that the Arduino Extensions use. These are used to eliminate a parameter that would otherwise have to be passed to every block. Go to the project properties (the wrench near the upper left hand corner) and select the ‘Variables’ tab. Press the ‘Add’ button at the bottom of the screen, set the name to ‘ExMusicAddress’ and its type to Numeric. Repeat for ‘ExLedAddress’. It should look like this:



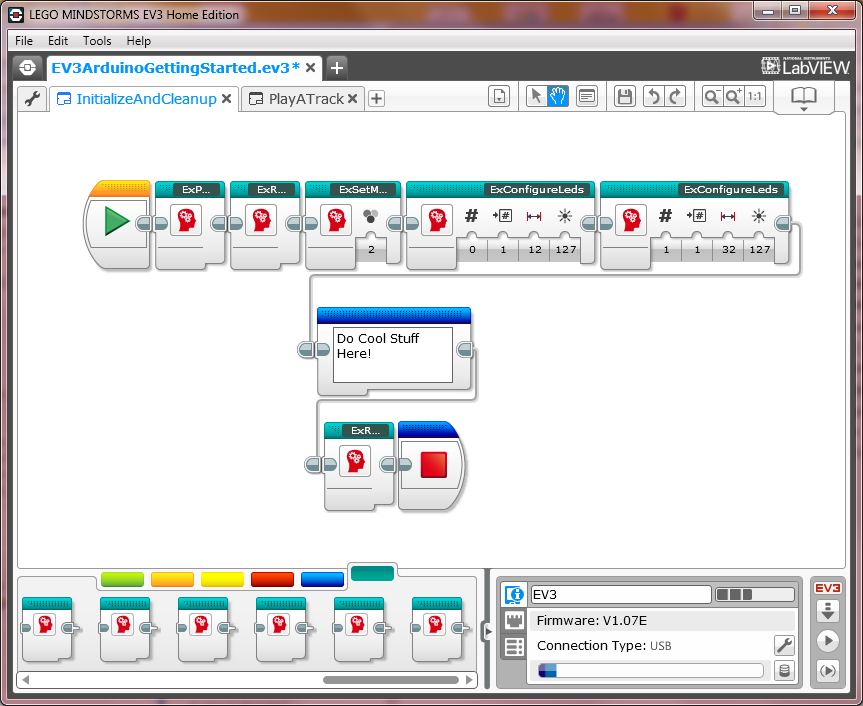
Next we need to import some blocks that will (hopefully) make it easier to work with the Arduino. These blocks will show up in the Teal group with other blocks that you define (called ‘My Blocks’). Go to the project properties (the wrench near the upper left hand corner) and select the ‘My Blocks’ tab. The list will be empty if it is a new project. Press the ‘Import’ button at the bottom of the screen and browse to and select the file ‘EvVArduinoExtFunctions.ev3s’. This should only be done once for each project, so if you see ‘PlayTrack.ev3p’ in the list, you shouldn’t do it again. At the right side of the screen is a column labeled ‘Show’ – you will want to remove the checkmarks for each line to avoid having it open them all up in the editor each time you open the project file.

Now we are ready to start using the Arduino Extensions. Start your project with the ExPowerOn and ExReset blocks as shown in the example below. The ExPowerOn block applies power to the Arduino Extensions module (if necessary). The ExReset block resets the Arduino to its default not-doing-anything state. If your program ever stops, then it is a good idea to place this same block right at the end (otherwise, the Arduino may just keep doing the last thing you told it to do). If your program never stops (which is quite common for an EV3 program), then don’t worry about it.

If you are going to use the sound capabilities, include the ExSetMusicMode block near the beginning of your program, passing a value of ‘1’ if you want to play notes and ‘2’ if you want to play tracks off of the SD card.

If you are going to use an LED strip connected to the Arduino, include the ExConfigureLeds block for each of the strips that you want to use (up to two). The ‘strip’ parameter is either ‘0’ or ‘1’ to identify the strip you want to configure, the ‘length’ parameter is the number of LEDs in the strip, the ‘frames’ parameter is the number of buffers you want to use (use 1 if you are not sure, more on this later), and the last parameter is a ‘brightness’ setting that is applied to every ExSetLed\* command. Starting at ‘128’ puts it in the middle and will give you the ability to raise or lower it as needed later.

In this example, it is using Track mode and two strips of LEDs, the first with 12 lights and the second with 32 lights.



## Playing Tracks

First thing that you have to do is have a ExSetMusicMode block with a parameter of ‘2’ in your program before you want to play the file. If you are only going to play tracks, then you can (and probably should) do it right at the beginning.

The hardest part about playing music files is getting the files on the SD card.

Probably the easiest to use are MP3 files. For this, simply rename the file to trackNNN.mp3 and place it on the SD card. The NNN is a decimal number padded with leading zeros. For example, track 16 would have a filename of track016.mp3.

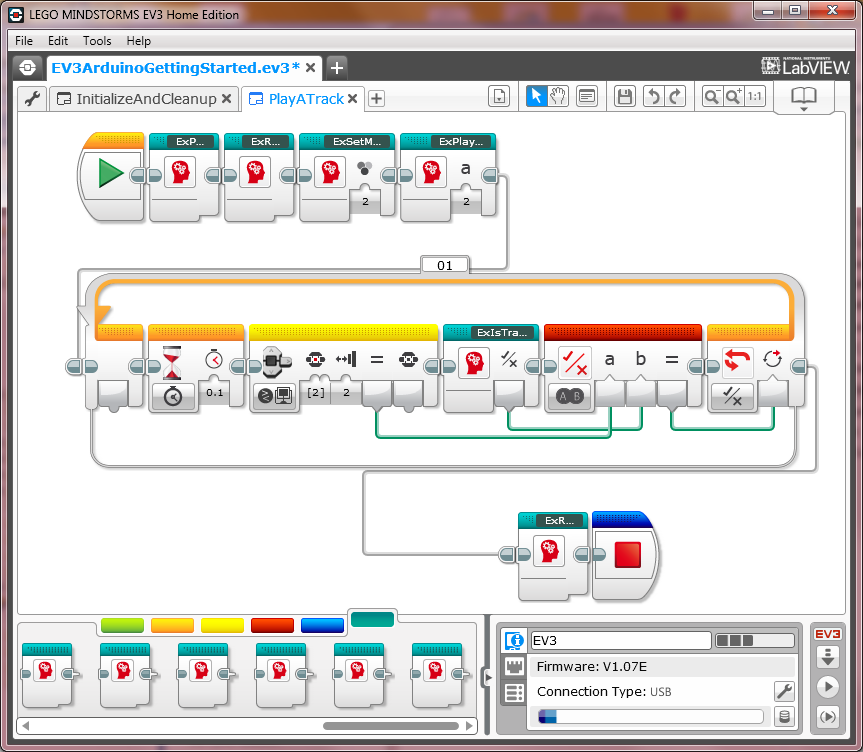
Once the desired track is on the SD card, you can play it with the ExPlayTrack block. The only parameter is the desired track number.

The Arduino will also play MIDI Type 0 files. Type 0 are files that contain only one track within the file. It may take some special steps to get a program like Garage Band to generate a MIDI Type 0 file. I found some MIDI files on the internet, and then had to run a conversion program on them to merge all of the tracks into a single track to make a Type 0 file. The good part about using MIDI files is that they are much smaller and take less of the Arduino’s time to play (leaving it free to do other things). But right now we don’t have much else for it to do, so it probably isn’t worth the trouble. A MIDI file should have a .mid extension on the SD card.

If the SD card has a trackNNN.mp3 and a trackNNN.mid and you tell it to play track N, it will play the .mp3. That is only because I had to pick one to look for first, and I choose .mp3.

For a short track, say a sound effect, it is probably fine to just play it and forget it. If you are playing a longer track and want to know when it is finished, you can use the ExIsTrackComplete block see if it is finished. It returns a logic output that is true if the track has completed.

Here is a simple program that plays track 2 and waits for it to finish or the user to push the center button. Your program will likely have more interesting stuff within the loop, like monitoring sensors or controlling motors.



## Led Strips

The Arduino Extensions module can support up to two Led strips. To use them, you need to start out with an ExConfigureLeds block near the start of your program.

Filling up the strips with color can be done in several different ways.

You can set an individual Led to a color using the ExSetLed block. Repeat this for the desired Leds and then use an ExShowLeds block to actually light them up.

You can set a range of Leds using the ExSetLedRangeFade block. This can fill anything from a few Leds to the entire strip with a single command. The color fades evenly from the color specified for the start Led to the color specified for the end Led.

If you are setting each Led to a specific color, it may be easier to put the desired colors into a numeric array and pass them to a ExSetLedRangeArray block. The array should contain three numbers for each Led to be set – a red, green, and blue value. If the array doesn’t fill the entire strip, you can tell the block where to put the data from the array.

Once you have the Leds set to the desired colors, there are some things you can do with them. One is that you can shift them left or right using the ExRotateAllLeds to move them around (since moving lights are much more cool then stationary lights). If you want them to keep moving and don’t want to have to keep telling them to move, use ExWalkAllLeds. With this block, you tell it how far to ‘rotate’ (as above) each time and how often to do it. Then they will keep moving until you tell them to stop – which you do by specifying a ‘count’ of zero.

# Limitations

The following are some somewhat arbitrary limitations that are imposed for various reasons. Most of these seem to be outside the range of normal use, but are listed here in case someone is trying something outside of ‘expected’ use and encounters problems.

The number of Leds in a strip is limited to 127. This is due to the constraint that we can only use the range of 0..127.

The number of frames for a strip is limited to 63. This is due to the way that I pack the parameters and the constraint that we can only use the range of 0..127 for bytes sent to the Arduino.

The total number of Leds is limited to about 300 on the ProMicro due to the amount of memory available. The total number is counted as the sum for each strip of the length of the strip times the number of frames allocated. So running one 32-Led strip with 8 frames counts as 256 Leds.

The track numbers are limited to the range 0..127. This is due to the constraint that we can only use the range of 0..127.

Intervals for the Walk blocks are limited to the range of .02 to 2 seconds in steps of (about) .02 seconds. This is due to the constraint that we can only use the range of 0..127 and I’m using units of 16 msec.

The count for the ExWalkLedFrames block is limited to 63. This is due to the way that I pack the parameters and the constraint that we can only use the range of 0..127 for bytes sent to the Arduino.

# Block Reference

## General Blocks

### ExPowerOn

#### Parameters

None.

#### Notes

This should precede the use of any other blocks.

### ExReset

#### Parameters

None.

#### Notes

This should immediately follow the ExPowerOn block. Why is it separate? Well, because it is sometimes helpful on its own.

This block will set the two variables ExMusicAddress and ExLedAddress both to values of 4 and 5, respectively, unless they are already set to a valid configuration.

## Music Blocks

### ExSetMusicMode

#### Parameters

* mode
  + 0 – turns off music.
  + 1 – selects Note mode in which you can play individual musical notes.
  + 2 – selects Track mode in which you can play entire tracks from the SD card.

#### Notes

This must precede any calls to play notes or tracks.

You can change the music mode as often as you like, but it was sort of expected that any project would pick one and use it throughout. There is sometimes noise generated on the audio outputs during the transition.

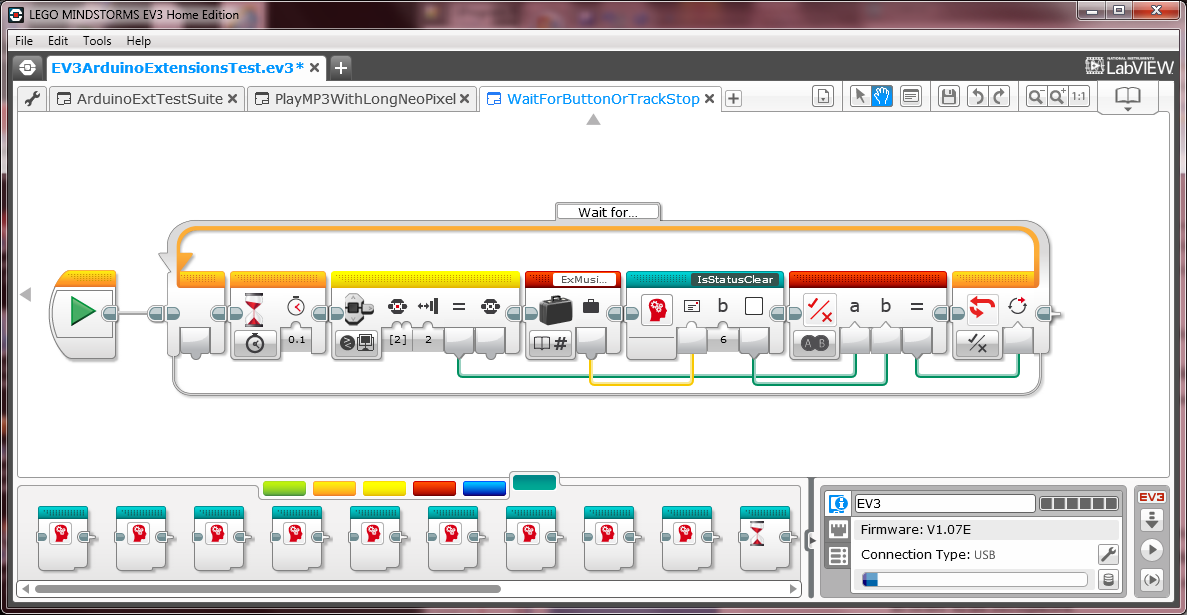
### ExPlayTrack

#### Parameters

* track – which track to play. The desired track must be stored in the root directory of the SD card and be named ‘trackNNN.xxx’. ‘NNN’ is the three digit track number (with leading zeros, if necessary). ‘xxx’ is the extension, either .mp3 or .mid (for MIDI files). MIDI files must be Type 0 in order to be recognized.

#### Notes

If you want to know whether playback has completed, use the ExIsTrackComplete block. Below is a block I created that waits for the track to stop playing or the center button to be pressed.



Starting playback of a track while one is already in progress will stop the prior playback and start the new one.

### ExIsTrackComplete

#### Parameters

none

#### Outputs

* result – the output is logical true if the track is complete, and false otherwise.

#### Notes

I opted to phrase the query as ‘complete’ because that is what seemed like the most natural question that a user might ask because they want to do something when it is finished.

### ExStopTrack

#### Parameters

none

#### Notes

This stops the playback of any track in progress, if any. It is fine to call it when no track is playing (or has just finished) if you just want to make sure the music has stopped.

### MIDI

This block has a number of variants. In its most basic form, you basically get to send raw MIDI commands that get passed on to the VS1053. In the higher forms, you get to select from among available instruments and start/stop notes.

If you want to get into this, the VS1053 data sheet is likely to be helpful. As is the MIDI specification.

Sorry, out of time to write about this section. There are a number of examples showing how to use the MIDI synthesizer.

## Led Blocks

### ExConfigureLeds

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* frames – the number of frames to allocate.
* length – the number of LEDs in the strip.
* brightness – the maximum brightness of the LEDs. All Set\* commands are scaled by this value.

#### Notes

This must precede any call to the ExSetLed\* functions.

The length of the Led strip is limited by the amount of power available and the amount of available RAM. A strip requires 3-bytes per LED. The Arduino currently has about 1K of RAM available, so it should be able to handle about 300 Leds split between the two strips.

Allocating multiple frames allows you to have multiple Led patterns setup that you can then quickly switch between, or have the Arduino walk through them at a specified interval. Support for frames isn’t complete, so using them is a little challenging. Sorry about that.

### ExShowLeds

#### Parameters

* strip – the Led strip to operate on, 0 or 1.

#### Notes

When you use ExSetLed\* blocks, the color of each Led is stored in memory on the Arduino. The ‘show’ actually takes the colors and sends them to the Led strip. The ExSetLed\* commands do not automatically do a show in order to allow you to perform multiple ExSetLed\* commands in order to get the strip into the desired colors before sending the colors out to the Leds at the same time.

Those blocks that operate on the entire strip generally include a show. For example, the ExSetLedBarGraph, ExRotateAllLeds and ExWalkAllLeds blocks always include a show.

### ExSetLed

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* index – the index of the Led for which to set the color.
* R, G, B – the color of the Led at ‘index’.

#### Notes

Hopefully you already know how to mix red, green, and blue to make any color that you want. White is (255, 255, 255), black is (0,0,0), shades of gray have equal amounts of each (N,N,N), and all other colors are various combinations of the three.

### ExSetLedRangeFade

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* start – the start index at which to start filling Leds.
* startR, startG, startB – the color of the Led at ‘start’.
* end – the end index (inclusive) at which to stop filling Leds.
* endR, endG, endB – the color of the Led at ‘end’.

#### Notes

Each color, red, green, and blue, are gradually faded between the start and end Leds.

For example, setting Led 0 to (255,0,0) through Led N-1 to (0,255,0) where ‘N’ is the length of the strip will fill the strip with colors fading from full red at one end to full blue at the other end.

### ExSetLedBarGraph

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* length – the length of the bar graph to create, in Leds.
* r, g, b – the color of the Leds.

#### Notes

Fills the Led strip with a bar of the specified length and color (starting at Led 0). The rest of the strip is cleared. This command includes a show.

### ExSetLedRangeArray

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* start – the first position in the Led strip to write to.
* array – the colors to place into the Led array.

#### Notes

The array must be numeric and contain three entries for each Led that you would like to set – the red, green, and blue values. The function will set the color of array.Length / 3 Leds, starting with the one at ‘start’.

This function is useful if the data that you want to place in the Led strip is not a uniform pattern. It is generally easier to setup the array than to create individual ExSetLed blocks for each Led in the strip.

### ExRotateAllLeds

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* count – the number of steps to rotate the Leds.

#### Notes

Positive ‘count’ shifts the Leds toward zero, and negative ‘count’ shifts the Leds away from zero. It is a ‘rotate’ in the sense that Leds that fall off the end roll back onto the other end to fill in the spaces vacated by the shift.

Note that a count of length-1 has the same effect as a count of -1.

This command includes a show.

### ExWalkAllLeds

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* count – the number of steps to rotate the Leds.
* interval – the time interval between rotations of the Leds, in seconds.

#### Notes

See notes for ExRotateAllLeds.

This blocks sets up the Arduino to do a rotate of all the Leds by ‘count’ steps every ‘interval’ seconds.

If count or interval is zero, then the command stops any walk currently in progress.

### ExWalkLedFrames

#### Parameters

* strip – the Led strip to operate on, 0 or 1.
* count – the number of Led frames to play.
* bounce – true if you want it to reverse direction when it reaches an end, false if you want it to start back at the beginning each time it reaches the end.
* interval – the time interval between frames, in seconds.

#### Notes

You cannot combine both ExWalkAllLeds and ExWalkLedFrames (because I’m not sure what that would look like).

## Low Level Blocks

Hopefully you don’t have to use these. They are primarily used in the implementation of the blocks listed above.

### SendCmdAndWait

#### Parameters

* address – the Arduino address to for which to check the status.
* cmd – the command to send.
* data1 – the parameter to the command.

#### Notes

This is a low level function that allows you to send commands directly to the Arduino. When available, it is better to use the functions provided specifically to do what you are trying to do.

For ‘address’, use the value of the variable ‘ExMusicAddress’ or ‘ExLedAddress’.

### IsStatusClear

#### Parameters

* address – the Arduino address to for which to check the status.
* bit – the bit number of interest.

#### Output:

* result – the output is logical true if the bit is clear, and logical false if the bit is set.

#### Notes

For ‘address’, use the value of the variable ‘ExMusicAddress’ or ‘ExLedAddress’.

The status bits are as follows:

* 0 – busy, set if the Arduino currently processing a command or has commands waiting in its queue.
* 1 – Note mode, set if Note mode is active.
* 2 – Track mode, set if Track mode is active (whether or not it is playing).
* 3 – not used.
* 4 – MMB detected, set if the AdaFruit Music Maker Breakout board (or shield) was detected. More specifically, it is set if it detected a VS1053 on the SPI bus at the specified chip select.
* 5 – SD detected, set if an SD card was detected in the card slot.
* 6 – playing, set if the Arduino is currently playing a track.
* 7 – true, always set to indicate this byte is a status readback.

# Compiling the Ardiuno

## Getting Ready

The Ardiuno Extensions code was originally developed and built using the Arduino 1.6.

The code requires several libraries that do not come with the standard installation.

* SimpleTimer – this is a library in the public domain. I got it from the Arduino Playground.
* Adafruit\_VS1053 – this is a library provided by AdaFruit to support their Music Maker Shield and Music Maker Breakout Board.
* Adafruit\_NeoPixel – this is a library provided by AdaFruit to support their line of Led strips they call NeoPixels, but it has been modified to add some additional capabilities. Specifically, support for rotation of the strip, setting a range of pixels with linearly interpolated colors, support for multiple frames, and the ability to clear a strip without allocating memory for it (i.e. a ‘show’ that writes all zeros).
* SD – this is a standard library, but it has been modified to eliminate functions that we are not using in order to save code space. Specifically, it eliminates all support for writing to the SD card. So be careful if you are building other Arduino projects that actually want this capability.

## Building

The EV3ArduinoExtensions file starts with these lines.

// Arduino Uno with the AdaFruit Music Maker Shield

#define UnoWithAdafruitMms 0

// Arduino Uno with the SparkFun Musical Instrument Shield

#define UnoWithSparkFunMis 0

// Pro Micro with the AdaFruit Music Maker Breakout board

#define ProMicroWithAdaFruitMmb 1

// Pro Micro alone

#define ProMicro 0

Set one of them to '1' to select that configuration to build for. These each represent one of the configurations that we are currently using. More can certainly be added as desired. All assume the EV3 interface (since that is the whole reason this code exists). Any can have either servos or NeoPixels attached as pins allow.

You must also select the appropriate Arduino flavor under Tools/Board. It is Arduino Uno for the first two, and SparkFun Pro Micro 5V/16Mhz for the last two.

## Memory Conservation

The Arduino Uno and Arduino Pro Micro are great little micro controllers, but they are both very resource constrained. The Uno has 32K of program space (yes, that is a K for Kilobytes) and 2K of RAM, while the ProMicro has 28K program space and 2.5K RAM. A fair amount of space is taken up by the libraries that are included to support the features and over which we have limited control (though some, as you will see). The Arduino build just barely fits when supporting the desired features. There are some memory conserving steps that have been taken.

### Features

First is the ability to compile in various features based on the particular configuration desired. The main (and only) source file is conditionally compiled based on the defined features for that configuration. Each of the features is, in theory, completely independent of the others. A feature includes the libraries needed to support it, the data structures and code to implement it, and the commands supported from the EV3. There are currently four of them that can be independently included or excluded from the build.

* featureSynth – this is the ability to control the VS1053 in MIDI synthesizer mode where you send individual notes.
* featureCodec – this is the ability to control the VS1053 in CODEC mode. CODEC stands for enCODe/DECode and refers to encoding and decoding music files (like MP3). It requires the SD card as well since that is where the music files are read from. This is (by far) the most expensive feature in terms of memory resources.
* featureServos – this is the ability to control two servos in simple repetitive motion.
* featureNeoPix – this is the ability to control two strips of NeoPixel Leds. These strips have a simple three wire interface, one for power, one for ground, and one for data. Each Led has its own very little computer that latches in its data and then passes everything else on to the next one. This allows a very large number of Leds to be controlled with only three connections to the Arduino.

The features are generally set by the choice of configuration, but that can be changed if you need to free up space.

There are two definitions that are related to features but not defined as a feature itself. This is the ability to switch the VS1053 between CODEC mode and MIDI mode. To accomplish this requires that two pins be connected to the respective signals on the shield/board, and then the following two #defines set to the selected pins.

#define vs1053Reset 9

#define vs1053Mode 12

If you have both featureCodec and featureSynth defined, you probably want to define these also.

### Verbosity

VERBOSITY levels. The Serial.print facility is quite useful for diagnosing what is going on in the Arduino (in the absence of a full debugger), but all those calls take up a lot of space. VERBOSITY set to 0 eliminates all output. VERBOSITY levels 1 through 3 add increasing detailed output. The difference between VERBOSITY 0 and VERBOSITY 3 is about 4K bytes, so to get more information you are likely going to have to remove features.

When adding new diagnostic statements, please give some thought as to which level they belong at, and then use one of the \_printN and \_printlnN variants rather than calling Serial.print[ln]() directly.

* Use VERBOSITY 1 for only the highest level statements, like program identification and significant errors.
* Use VERBOSITY 2 for statements like program configuration and significant mode changes.
* Use VERBOSITY 3 for everything else, like debug statements, echo of command parameters, and progress messages.

### SD Library

Support for writing to the SD card has been removed (as mentioned above). This saved over 3K bytes of program storage. The Arduino linker is actually really good at not including any code that can’t actually be run, so even if a function is defined but never called, it doesn’t get included. But the interface to the SD card library is such that there are some functions, such as ‘open’ that operate for both read and write and result in all of the ‘write’ support being included even if you never call ‘write’ or open a file with ‘write’ permission. So getting rid of them required modification of the SD library in order to remove undesired link-time dependencies. It is done by setting the #define SD\_EXCLUDE\_WRITE 1 in the header file libraries/SD/utility/SdFat.h. To restore the ability to write to the SD card (say if you want to log data to the SD card), use #define SD\_EXCLUDE\_WRITE 0. Unfortunately, I can figure out a way to do this from within the project.

# EV3 <-> Arduino I2C Protocol

At the lowest level, there is a stream of bytes that pass between the EV3 and the Arduino, and this section describes what the meaning of that stream of bytes is.

The basic format is a single byte command followed by a specified number of data bytes for that command.

|  |  |  |
| --- | --- | --- |
| Command | Data / Parameters | Description |
| 1 | 1 byte  0 – mode. | For ‘mode’, valid values are 0 for none (turns off any active mode), 1 for Note mode, or 2 for Track mode. The expectation is that a given application will use one mode or the other, and that is why the switch must be performed explicitly. |
| 2 | 1 byte  0 – track. | Play the file on the SD card name ‘trackNNN.mp3’ or ‘trackNNN.mid’, where NNN is the parameter track expressed in decimal with leading zeros to make it 3 digits.  Does nothing if not in Track mode. |
| 4 | 1 byte  0 – volume. | Set the audio volume, from 0 (softest) to 127 (loudest). Can be used for both Note mode and Track mode. In Note mode, it sets the volume for all channels (there is no global volume). For individual channel control, use the MIDI blocks MIDI Cmd to send (0xB0 | chan), 0x07, volume. |
| 5 | 1 byte  0 – unused. | Stop track playback (if any).  Sorry for the unused parameter – there was a time when all commands were two bytes long. |
|  |  |  |
|  |  |  |
|  |  |  |
| 20 | 3 bytes  0 – strip, frames  1 – length  2 – brightness | Configure the specified Led strip.  ‘strip’ is 0 or 1 and contained in low bit of the first byte. ‘frames’ is in bits 6..1 of the first byte and represents the desired number of frames-1 (i.e. a value of 0 gives one frame, the default).  For ‘length’, it is the number of Leds in the strip, and a value of zero disables the strip.  For ‘brightness’, it is a global brightness value applied to all color values written to the strip. |
| 21 | 1 byte  0 – strip, frame | Show the Leds. When setting colors, they are generally just written to an internal buffer, allowing you to set them all before updating them all at the same time. The exception is commands that affect the entire strip, in which case they are immediately shown.  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1. |
| 24 | 5 bytes  0 – strip, frame  1 – index  2 – red  3 – green  4 – blue | Set the index and color for a multi-Led set. See command 25.  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1.  For ‘index’, the valid range is from 0 to the number of Leds – 1.  The color is specified as (red, green, blue). |
| 25 | 5 bytes  0 – strip, frame  1 – index  2 – red  3 – green  4 – blue | Set the color of the Led at ‘index’ to the color (red, green, blue). If command 24 precedes it, then it starts at the Led specified in command 24 and fills to the Led specified here, fading the colors gradually between the two.  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1.  For ‘index’, the valid range is from 0 to the number of Leds – 1.  The color is specified as (red, green, blue). |
| 26 | 5 bytes  0 – strip, frame  1 – unused  2 – red  3 – green  4 – blue | Set the color of all the Leds to the color (red, green, blue).  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1.  The color is specified as (red, green, blue). |
| 30 | 3 bytes  0 – strip, frame  1 – steps  2 – unused | Rotate the contents of the specified strip/frame by ‘steps’ toward the beginning of the strip. Negative numbers rotate away from the beginning of the strip. Values rotated off one end of the strip wrap around to the other end.  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1.  For ‘steps’ useful values are from –(N-1) to (N-1) where N is the number of Leds in the strip. |
| 31 | 3 bytes  0 – strip, frame  1 – steps  2 – interval | Rotate the contents as above, repeating it every ‘interval’ until told to stop.  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1.  For ‘steps’ it is as above. If zero, then any prior repeating rotate is stopped.  For ‘interval’ it is expressed in units of 16 msec to give it a reasonable range in a single byte. |
| 32 | 3 bytes  0 – strip, frame  1 – bounce, steps  2 – interval | Play the frames in the strip, advancing the frame every ‘interval’.  The ‘strip’ and ‘frame’ parameters are packed into the first byte, with ‘strip’ in bit 0 and ‘frame’ in bits 6..1.  ‘bounce’ is bit 6 of the second byte. Setting to 1 will reverse the order of playback when it reaches the end, while setting to 0 will wrap around to the first frame when it reaches the end.  ‘steps’ is bits 5..0 of the second byte and is the number of frames to play. If zero, then play frames until commanded to stop.  For ‘interval’ it is expressed in units of 16 msec to give it a reasonable range in a single byte. If zero, any prior frame playback is stopped. |
| 127 | None. | Restores everything to its default power-on state. This stops anything that the Arduino is doing, including any track that is playing, servos that are moving, and lights that are showing. |
| 128-255 | 2 data bytes | MIDI Synthesizer command.  Does nothing if not in SYNTH mode. |

Performing a read from the Arduino returns a status indicating its current state. See the reference for IsStatusClear for a description of the meaning of each bit in the returned byte. Note that a zero is returned if the Arduino fails to supply a byte to the I2C bus in time, so the reader should check that the most significant bit is set so it knows it got a real status value.

The Arduino does not care how you send the stream of bytes. A single command may be written a a single write over I2C, or multiple commands may be packed into a single write, or a single command may be sent as multiple writes. The only constraint on the last one is the Ardunio will only wait so long to receive the data to go with the command before discarding it.

There is currently a constraint that any command requiring more than three bytes must limit the range of values of each byte to 0..127. A problem that Steve Cogger is best able to describe.