SYNTHIAM

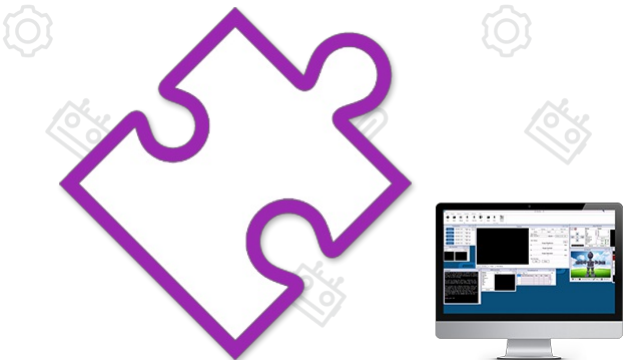
synthiam.com

Make an ARC Skill

Have a fantastic idea for a new skill in ARC to publish in the Technology Store? This is the tutorial which explains how to create a skill for ARC. This tutorial will help you create an example skill with two buttons which move a servo to get you started.

Last Updated: 4/7/2020

Overview



Summary In this tutorial, we will cover the following steps...

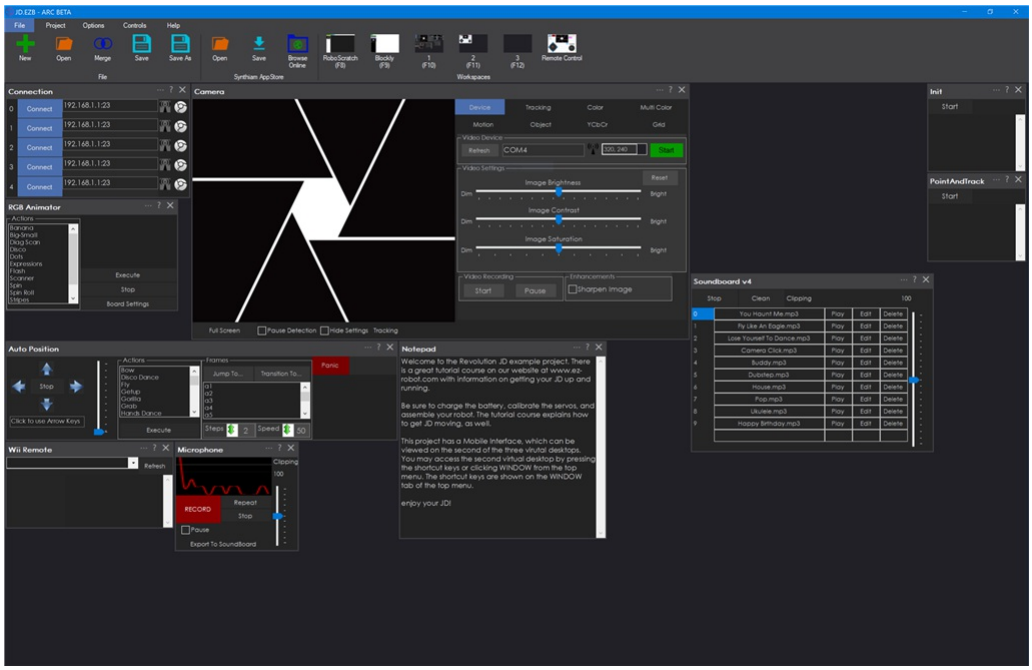
1. Download and install the latest ARC
2. Download and install Visual Studio
3. Create a new skill plugin Visual Studio project in ARC
4. Make your skill do amazing stuff
5. Upload the skill package to Technology Store and share with the community!

Skill Design Tip Skills can communicate with each other. This allows your skill to receive commands from other skills, programmatically.Â The way skills interact with each other is with scriptingÂ *ControlCommand()*Â script functions. To respond to ControlCommand() requests, your skill will most likely containÂ *SendCommand()* and *GetControlCommands()*Â override listeners. These will allow other skills to send instructions to your skill. Keep this in mind when designing your skill, as it's discussed in this tutorial.

Download ARC

ARC is the software which you will be creating your skill for. Your skill is a graphical library that runs within the ARC open framework. The open framework allows your skill to have the ability to use ARC's pre-built functions, rather than creating an entire program starting completely from scratch. This allows robot builders to install your skill from the technology store to add new features to their robot.

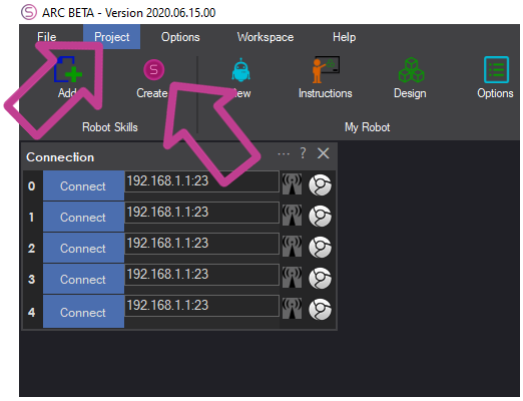
Download and install the latest version of ARC from here.



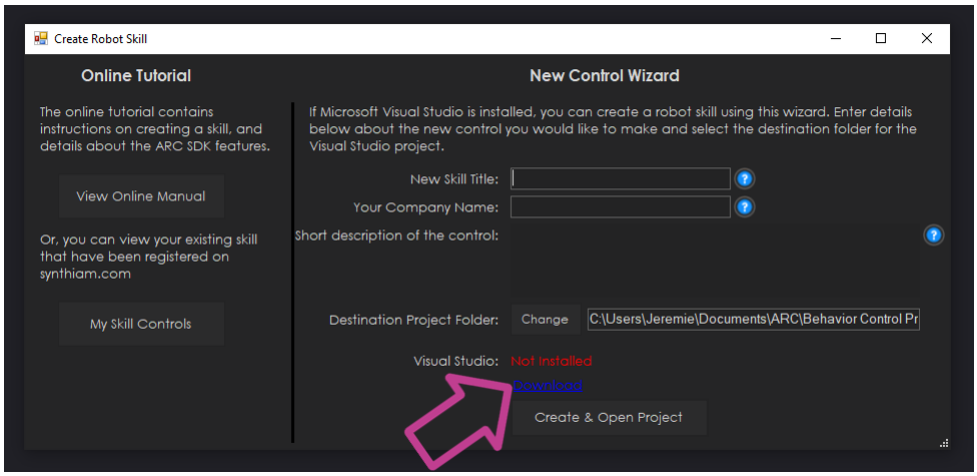
Install Visual Studio

Visual Studio is a software development IDE from Microsoft. It was once a very expensive software package for professionals, and now it has been released open-source and free! The community edition can be downloaded for free and gives you the power to create .Net applications. In this step, we'll use ARC to detect an existing installation of Visual Studio and download the latest.

1. Load ARC
2. Press **Project -> Create Skill**



ï»¿ 3) The **Create Robot Skill** window will check for an existing installation of Visual Studio. If there is not one detected, a *DOWNLOAD* button will be presented.



ï»¿ 4) Press the Download link if presented, or get theÂ *Community Edition* of Visual Studio by clicking here.

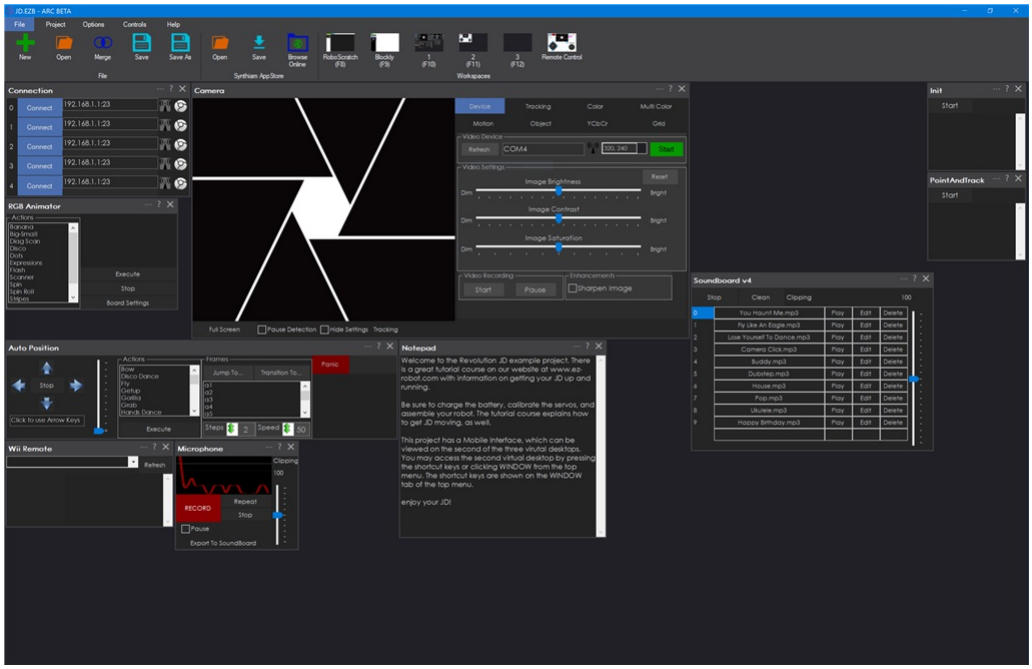
Dependency Ensure you have .Net 4.7.2 framework installed. If you keep up-to-date with Windows Updates and ARC runs without problems, then you must have .Net 4.7.2. Many programs are built with .Net 4.7.2 and therefore it's uncommon for a computer to not have it. So, if ARC works then you don't need to download and install .Net 4.7.2

Create Project

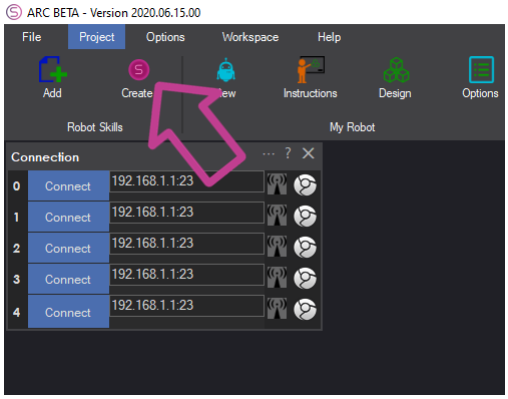
***Note:** If you are using C++ instead of C#, the setup is similar if you are experienced with creating managed C++ DLL libraries.

ARC will automatically generate an example project with details about your skill. To do so, first load the Create Robot Skill dialog box.

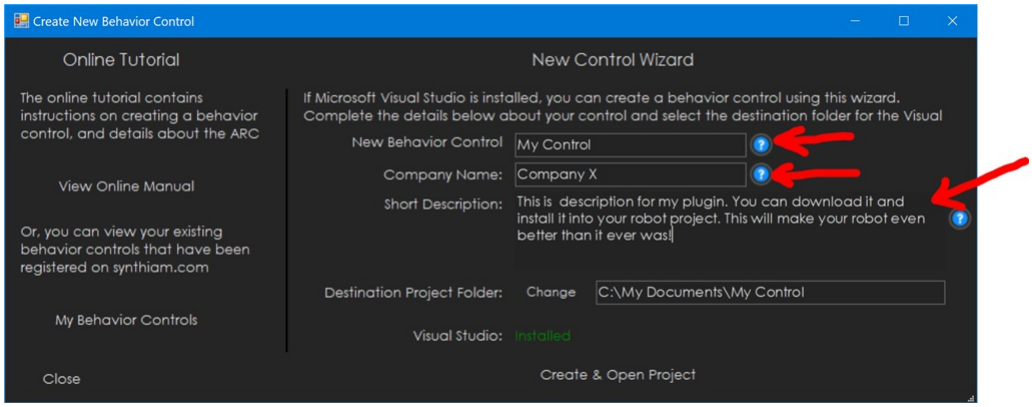
1. Load ARC



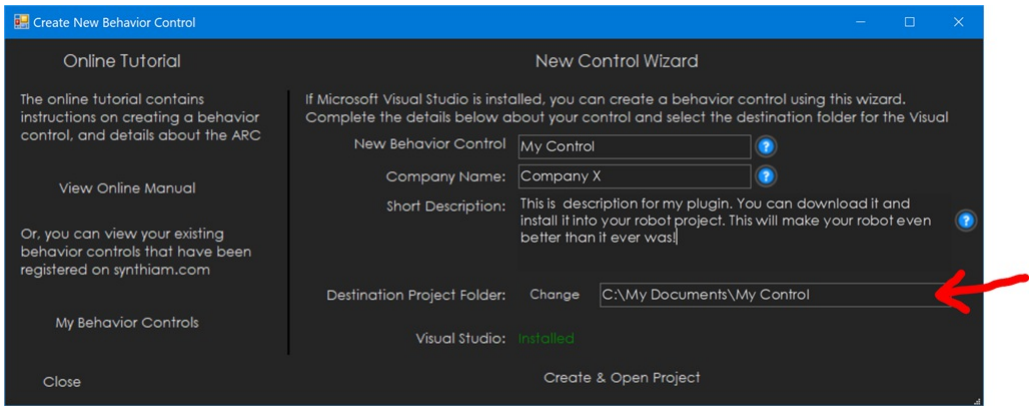
2. Press *Project* -> *Controls* -> *Create Control*



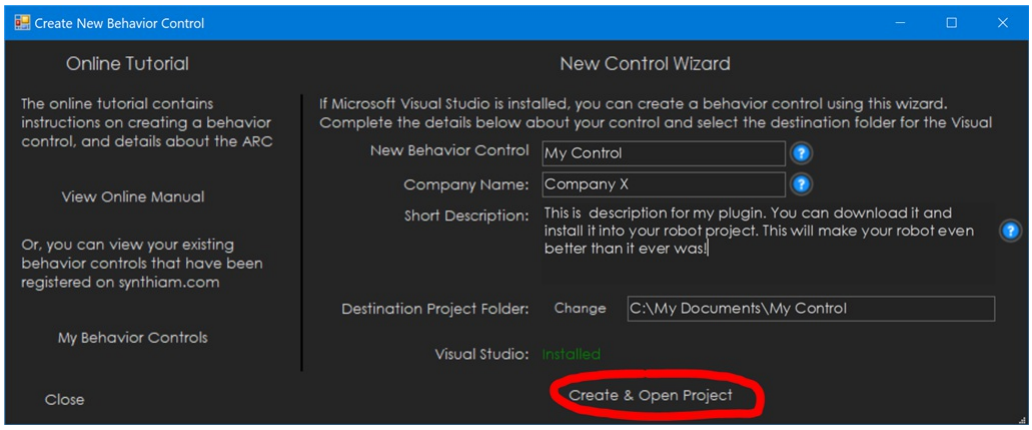
3. Enter the name of your new skill, company name, and a short descriptionÂ



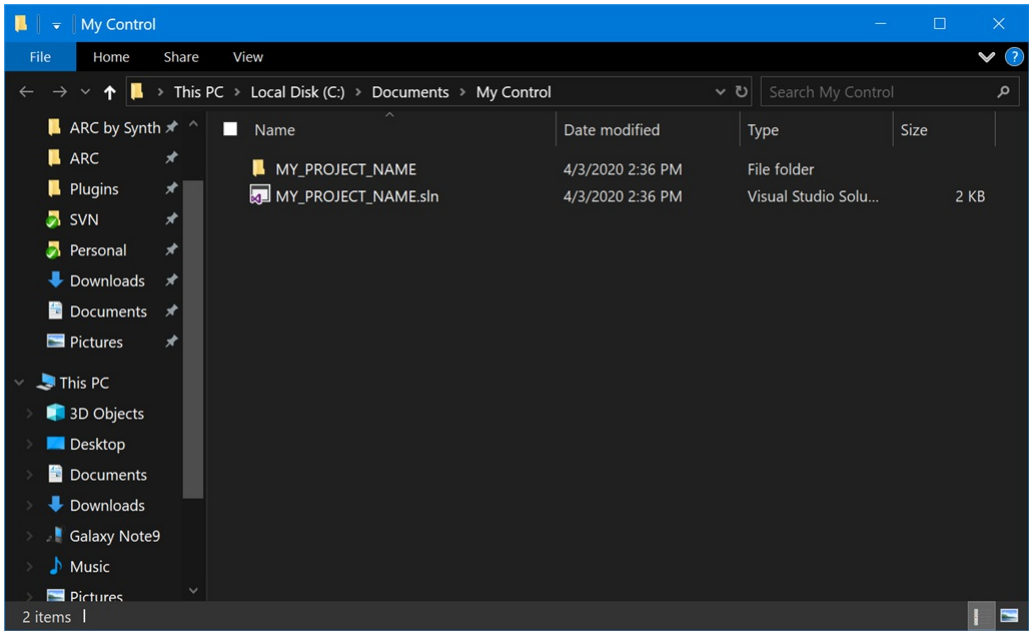
4. Select a location for your project. Otherwise, the default location is usually best to keep them easily accessible.



5. Press *Create & Open Project*



6. A folder will open with the location of your project solution. Double click on the SLN file to open the project in Visual Studio.

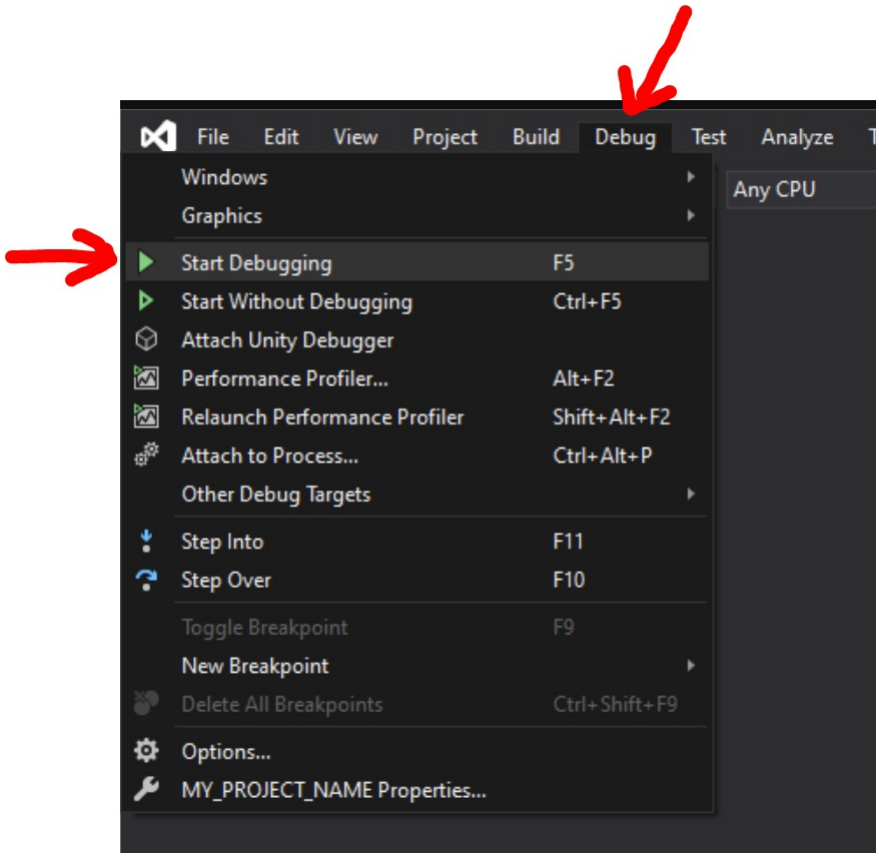


Example Project Overview

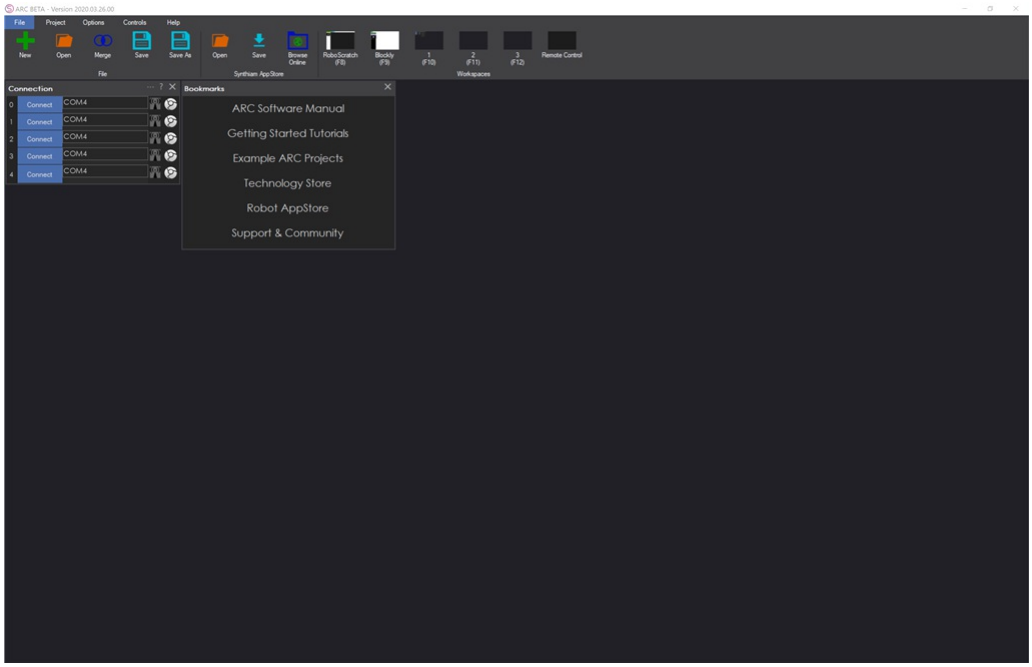
The example project will add the references and configuration necessary to begin developing.

Running The Example Project Let us first take a moment to load the example project in ARC and see it in action.

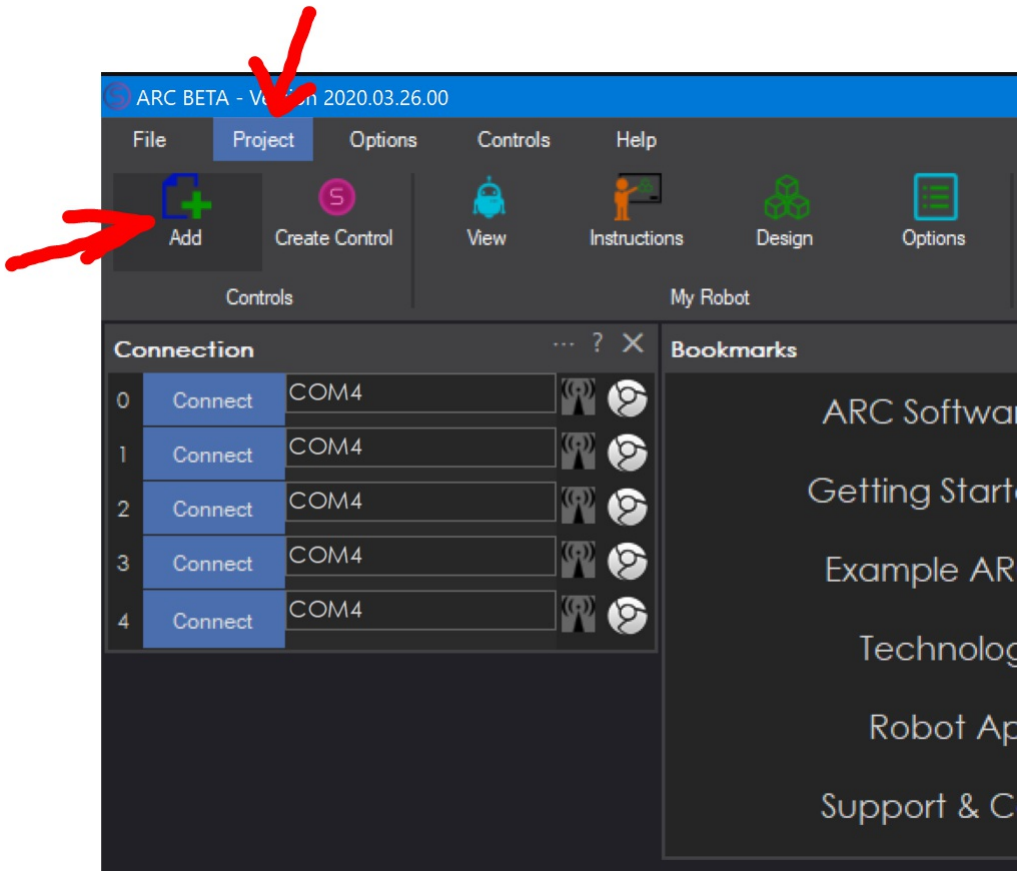
1. Shutdown any instances of ARC, as they aren't necessary during debugging.
2. Load the Visual Studio solution file for your plugin.
3. Press **F5** in Visual Studio, or select *Debug -> Start Debugging* from the top menu



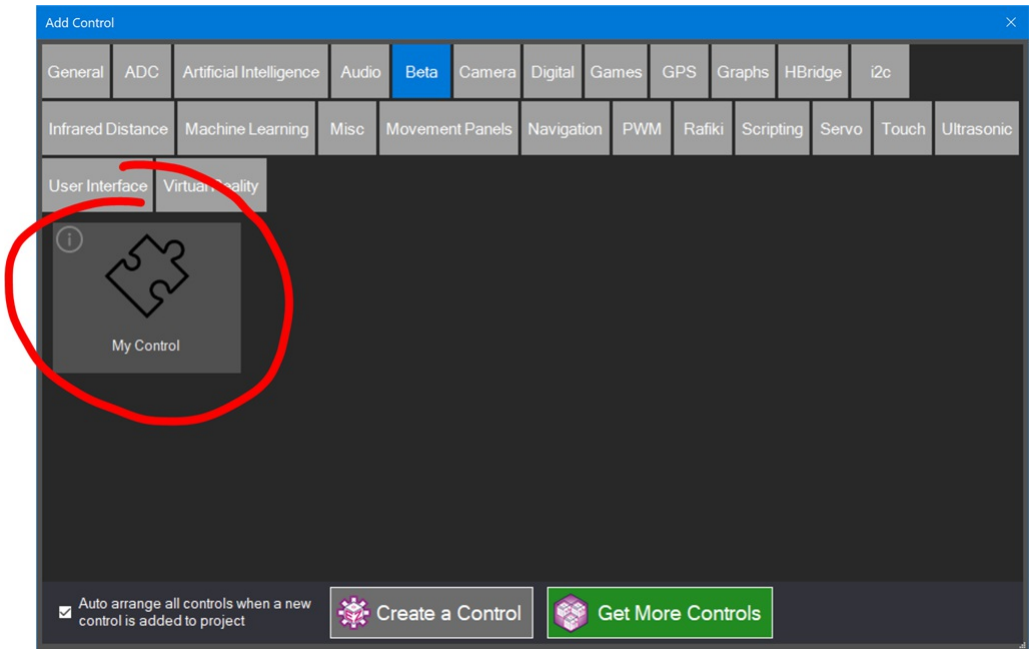
4. ARC will launch



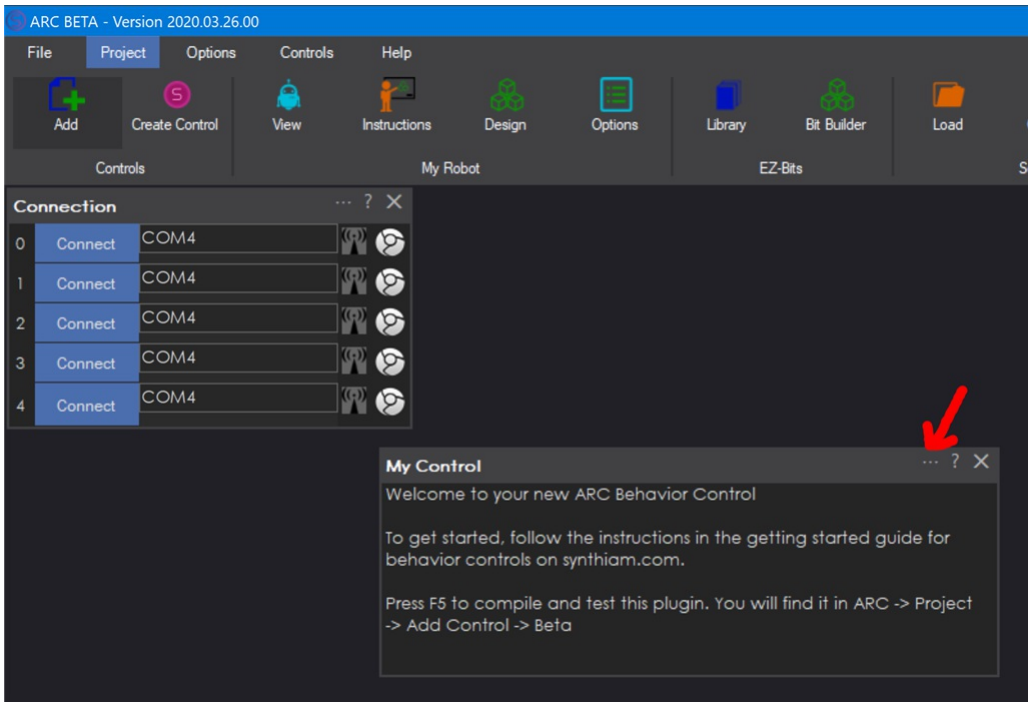
5. We'll now add your skill to the workspace. Press *Project -> Add Skill*



6. Navigate to the *Beta* tab, and your skill will be visible. Click the icon and the project will load.



7. View your new skill on the workspace. The example project even has a configuration menu configured. You can press the configuration option (3 little dots next to the X) to view the example configuration menu.

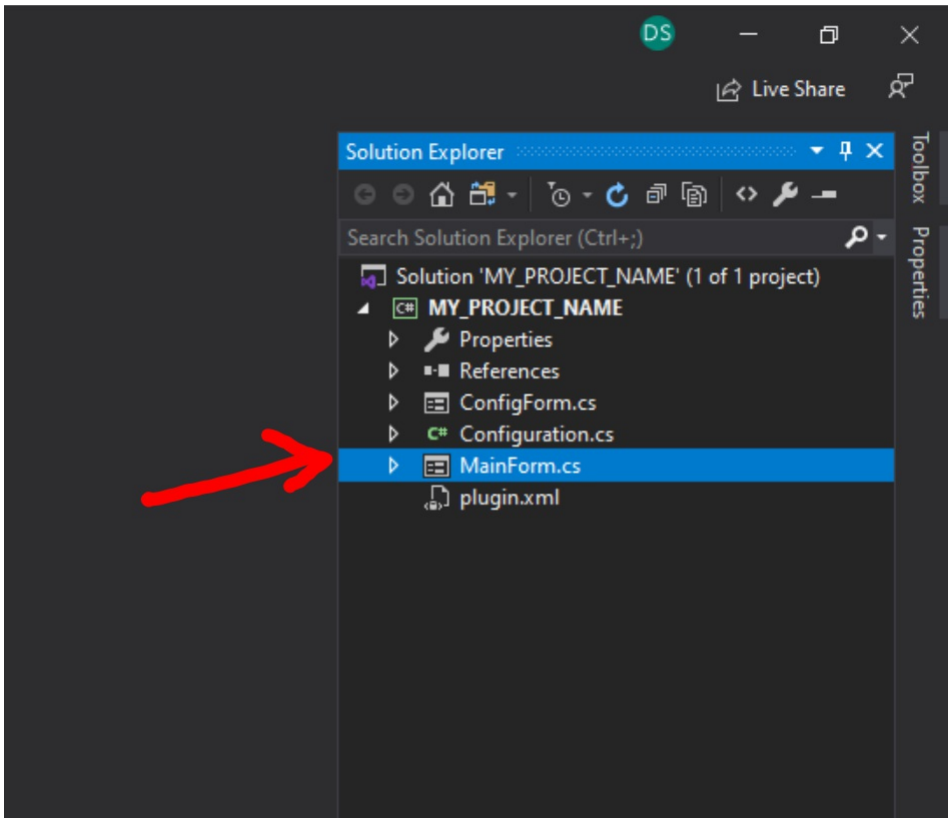


Notes Visual Studio is running as the debugger for the project. For experienced programmers, you can step through and debug the robot skill plugin in real-time.

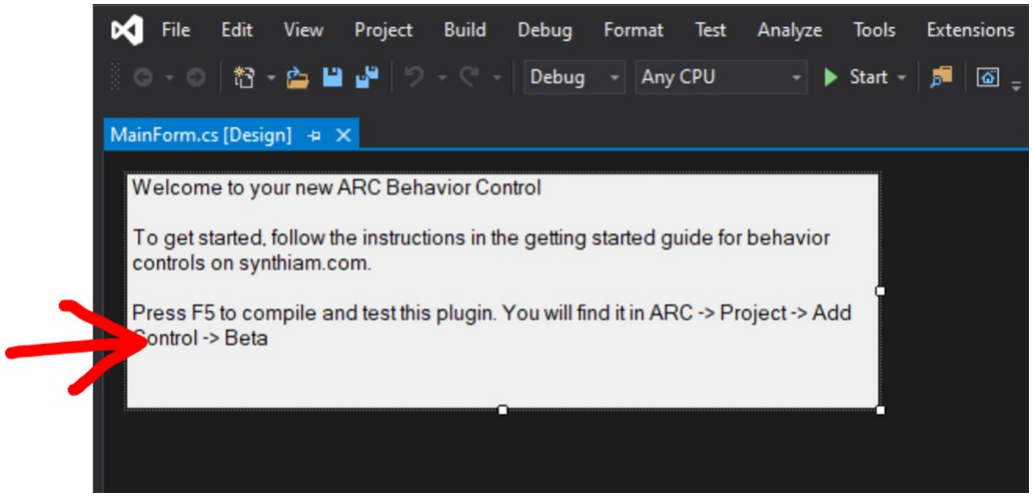
Add Some Buttons

Now that the **MainForm** has been created with the example project, it will be the form that users see when using your plugin. The form is currently only containing a label with some text, making it a boring skill. In this step we will add two buttons to move servos between different positions.

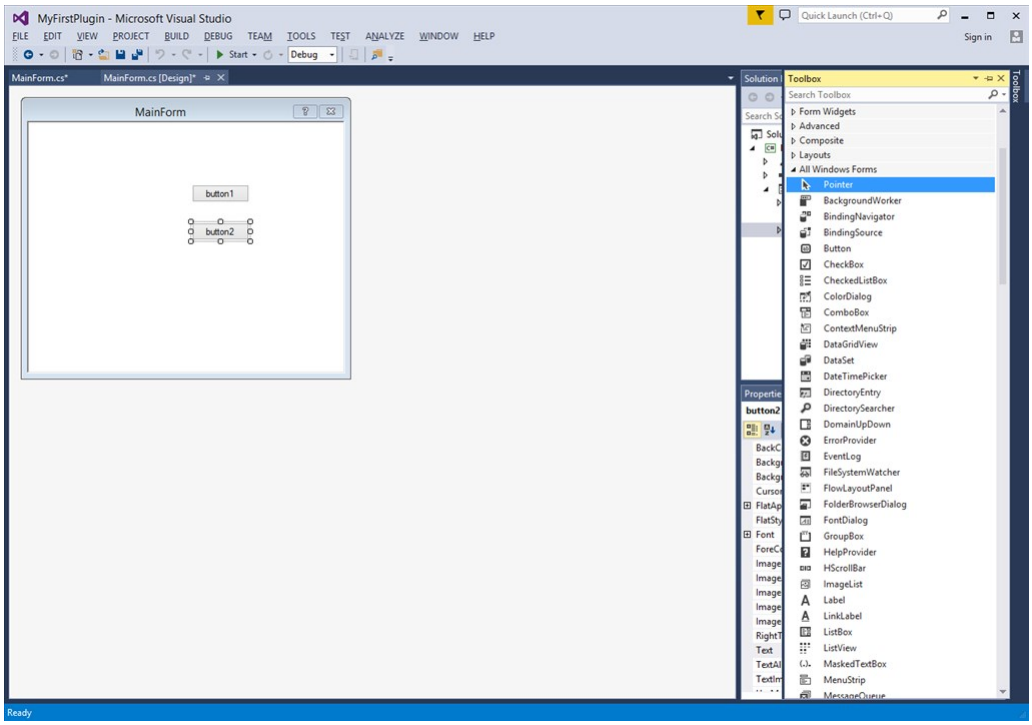
1. Locate the **MainForm** in the Solution Explorer and double click.



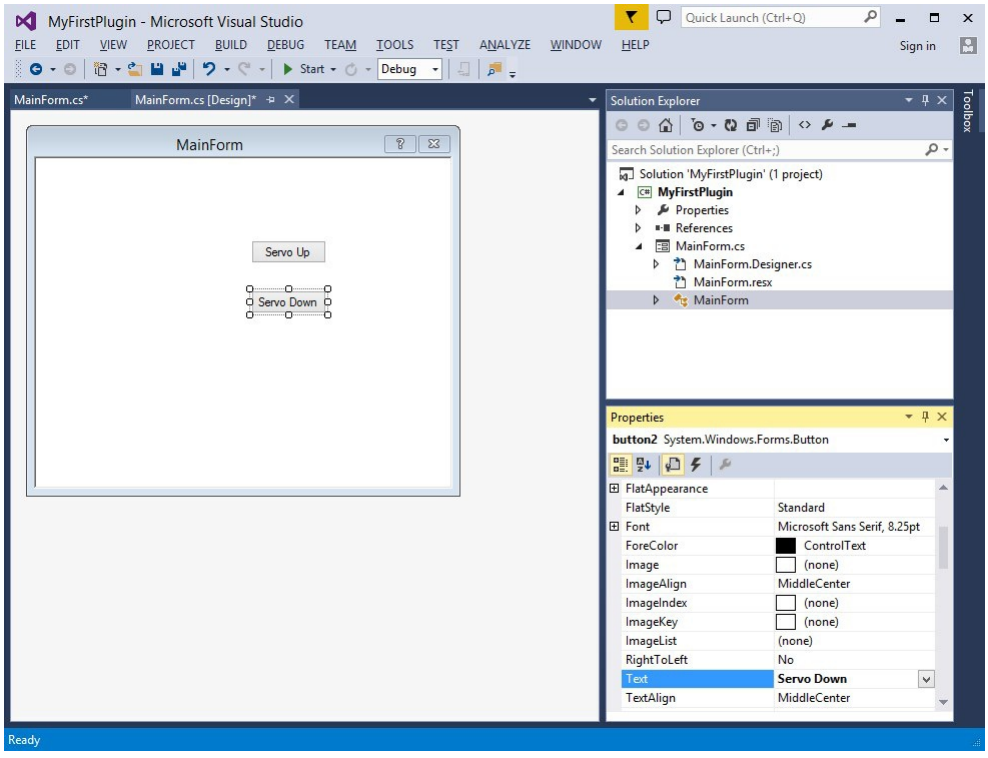
2. When the MainForm designer loads, click on the text and press Del to remove the label from the form. This will leave you with a blank form.



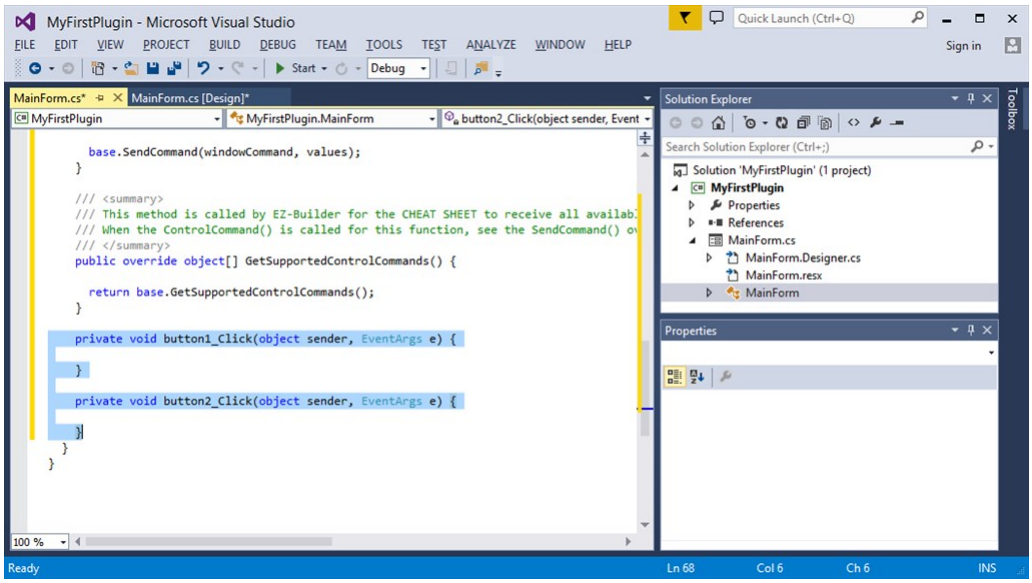
3. Locate the **Button** under *All Windows Forms* in the Toolbox. Drag two buttons anywhere onto your **MainForm**.



4. Give the buttons readable text that tells the user what they will do. In this tutorial example, we will be programming the buttons to move a servo between two positions. Click on each button and locate the **Text** field in the properties window.



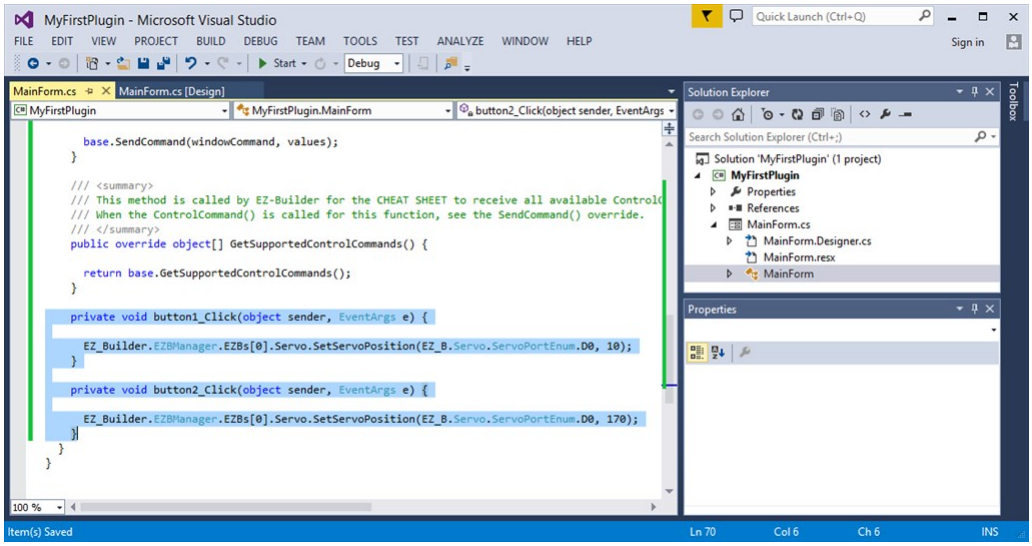
5. Double click on each button in the Designer and code will automatically be generated for the Click event of each button. This means that when a user clicks on the button, the code within the function will be executed. The functions are automatically inserted into your code when you double click on them from the designer.



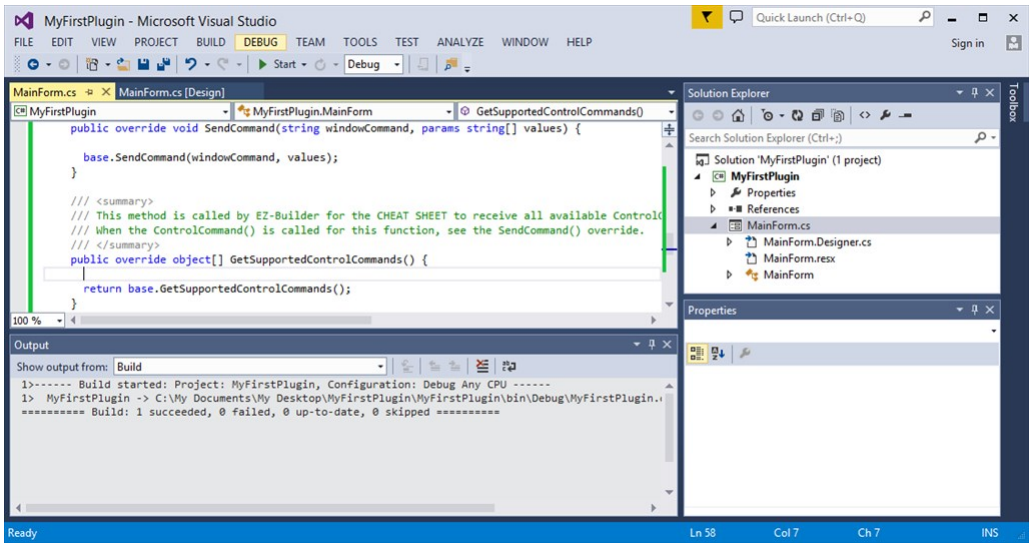
6. Insert code into each of the button click events to move a servo. The command to move a servo is located within an EZB class. Because ARC allows more than one EZB connection, the list of EZB's available is an array. It is safe to assume that the first EZB is used the most. Here is the code which will move the servo connected on the EZ-B port D0 between position 10 degrees and 170 degrees when the buttons are pressed. ``` private void button1_Click(object sender, EventArgs e) { EZ_Builder.EZBManager.EZBs[0].Servo.SetServoPosition(EZ_B.Servo.ServoPortEnum.D0, 10); }

   private void button2_Click(object sender, EventArgs e) { EZ_Builder.EZBManager.EZBs[0].Servo.SetServoPosition(EZ_B.Servo.ServoPortEnum.D0, 170); } ```

7. When your code has been entered, it will now look like this.



8. Let's compile your project to ensure there are no errors before continuing to the next step. Press **CTRL-SHIFT-B** and watch the Output window for any error messages. If everything compiles okay, you will see a similar message result to the screenshot below.



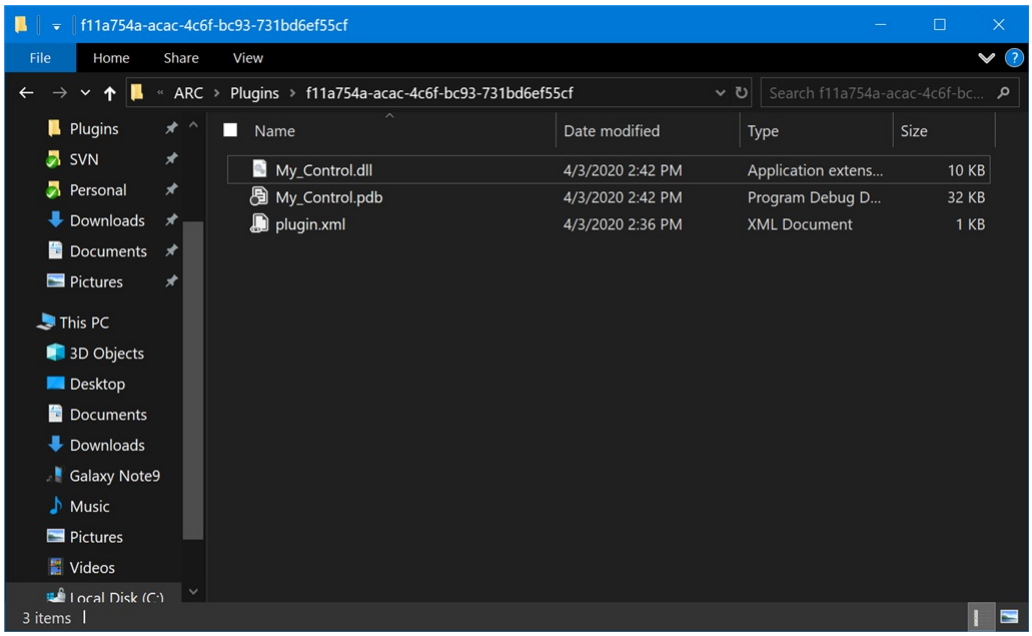
9. Assuming you have no errors and everything compiles fine, press F5, the project will compile and load ARC. Add the robot skill plugin to your workspace and test it out!

Publish Skill To Technology Store

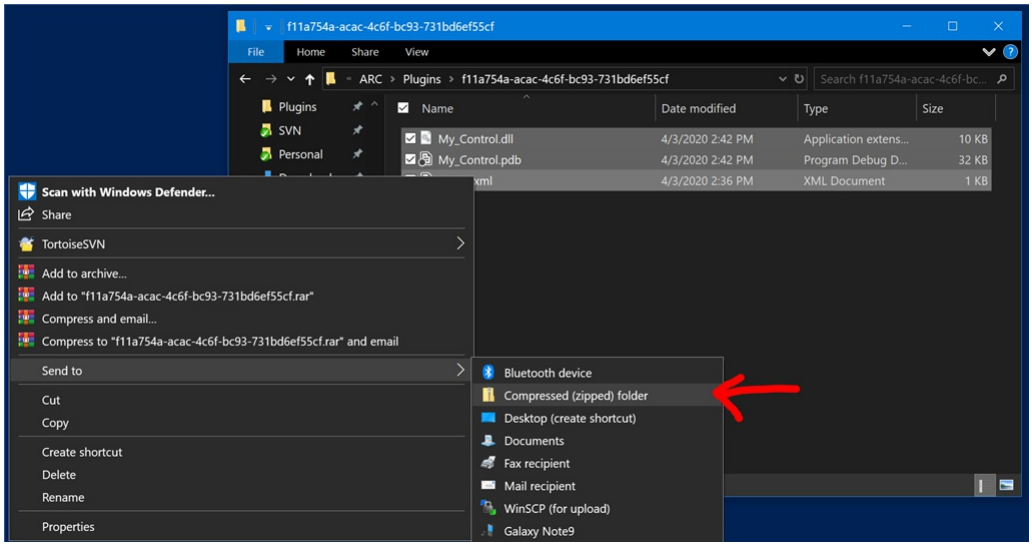
When you are ready to share your robot skill with the world, it will be published to the technology store on Synthiam's website.

Create Package

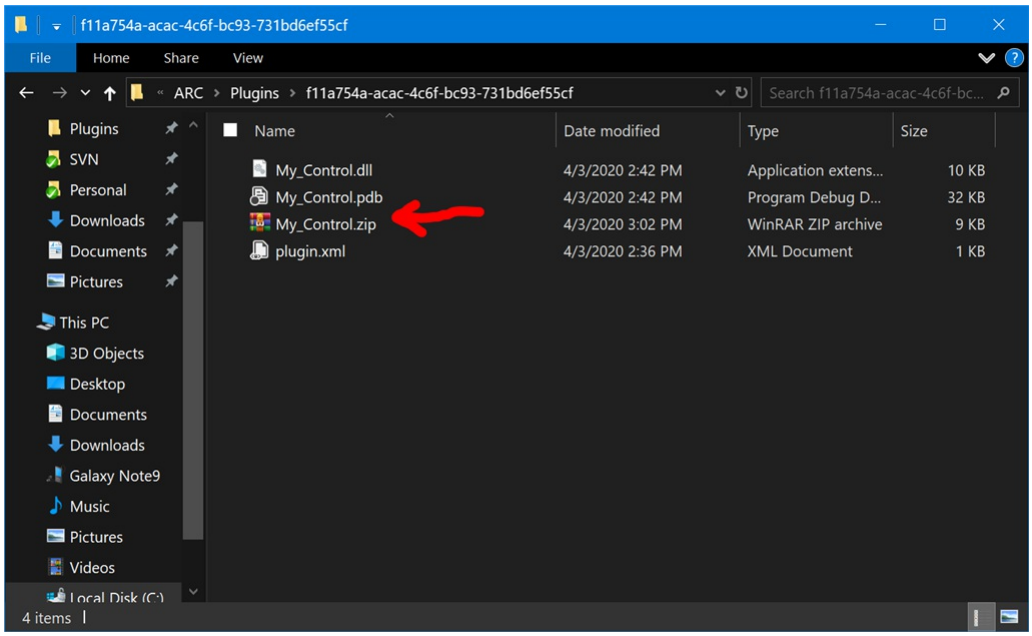
1. Build a fresh copy of your robot skill plugin in Visual Studio.
2. Navigate to the plugin folder. This will be located inÂ *C:\ProgramData\ARC\Pluginsï»¿<GUID>*. Where <GUID> is the guid of your plugin in the Plugin.XML file.



3. Select all files i[/i] and *Right-Click* with the mouse. Select *Send To -> Compressed (zipped) Folder*

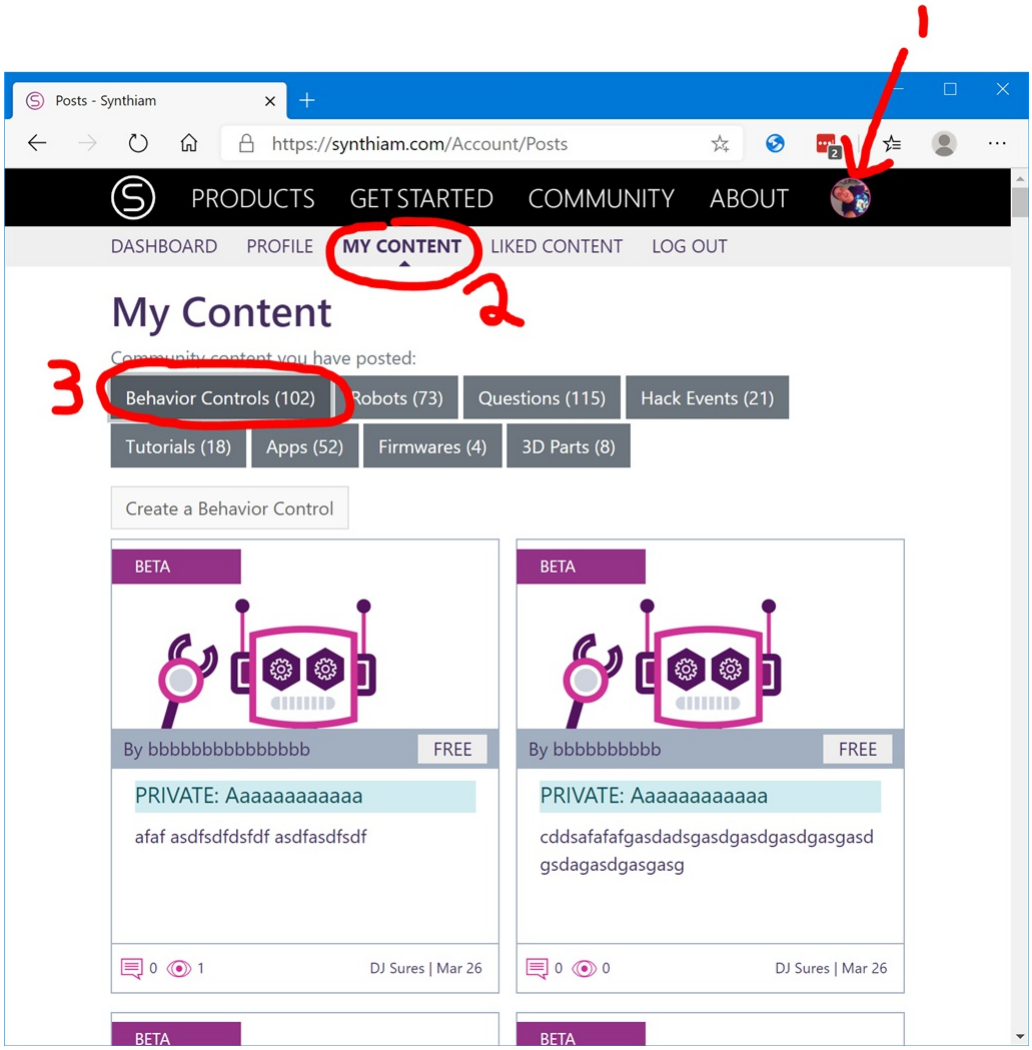


4) A .ZIP file will be created containing all of the necessary behavior control plugins and sub-folders. This is the file that will be uploaded to Synthiam.comÂ

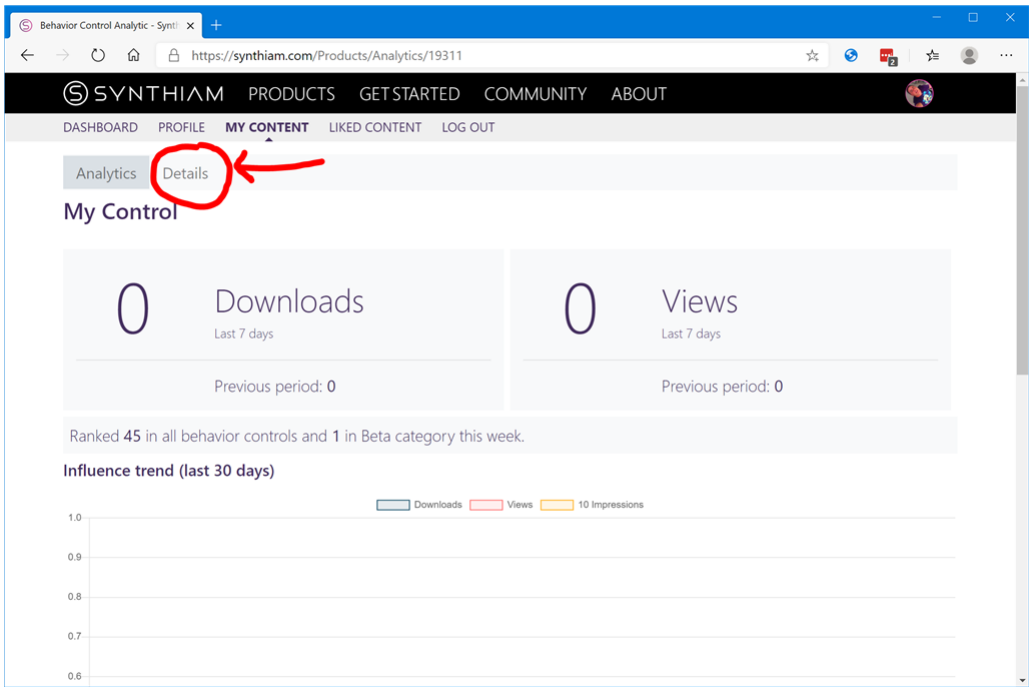


Upload Package To Synthiam The zip file created in the above step will be uploaded to the synthiam website.

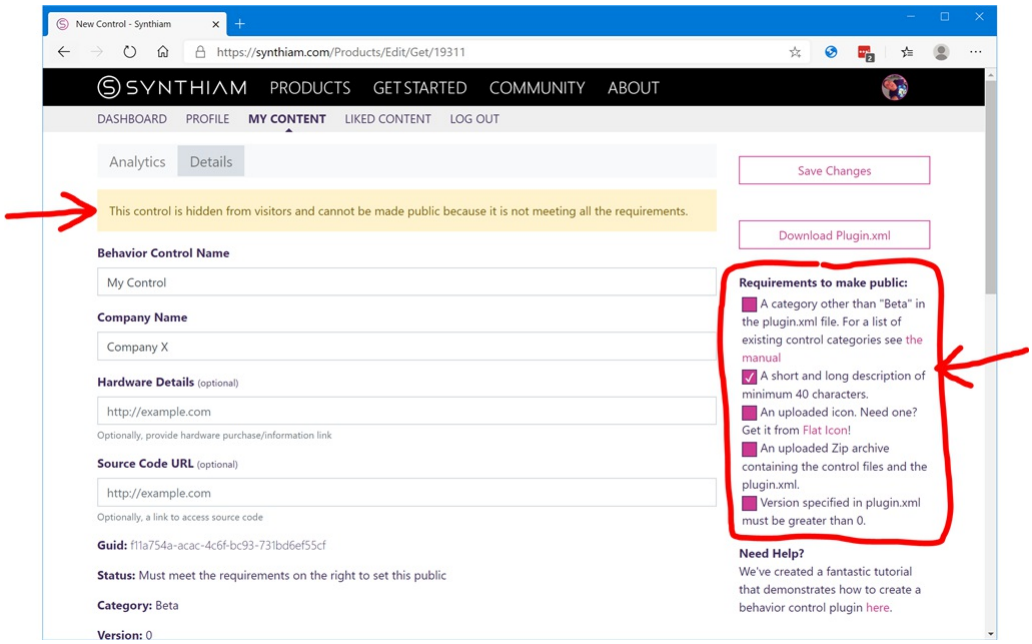
1. Visit synthiam.com and login.
2. Press the Account button on the top right.
3. Press the My Content sub-menu.Â



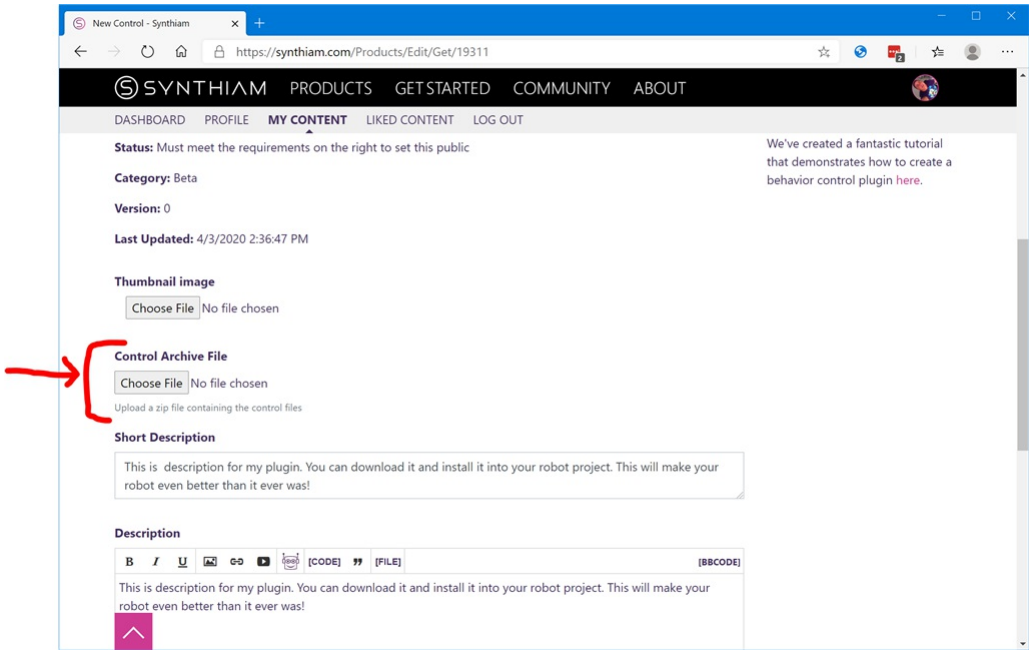
4. Locate your robot skill plugin by the title and click to select it. The skill statistic page will be displayed. When your skill is live, you may return here to view download and usage performance.Â Press the Details button.



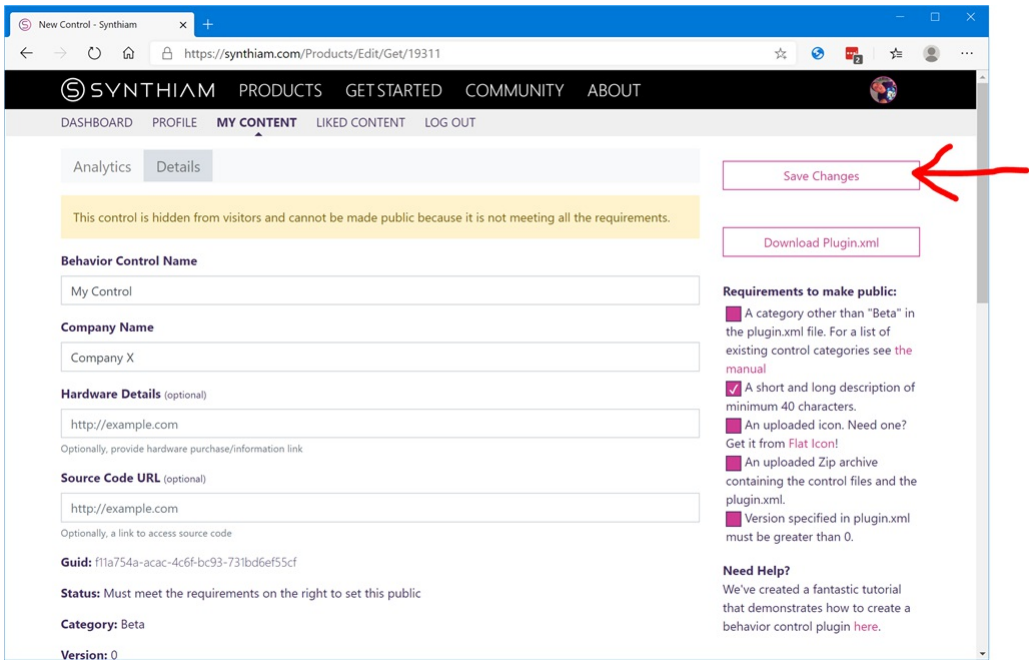
5. By default, the details page will introduce your skill as Private and display a number of requirements to make it public.



6. Scroll down and locate the Control Archive File option. This is where we will upload the skill Zip file package that was created earlier.



7. Scroll up and press Save Changes. The package file will upload.Â



Notes Before your robot skill plugin can be visible to the public, some requirements must be met. Review the list of requirements on the right of the details page. Once your requirements have been met, check the Public checkbox and your plugin will be published to the Synthiam Technology Store.

More Information

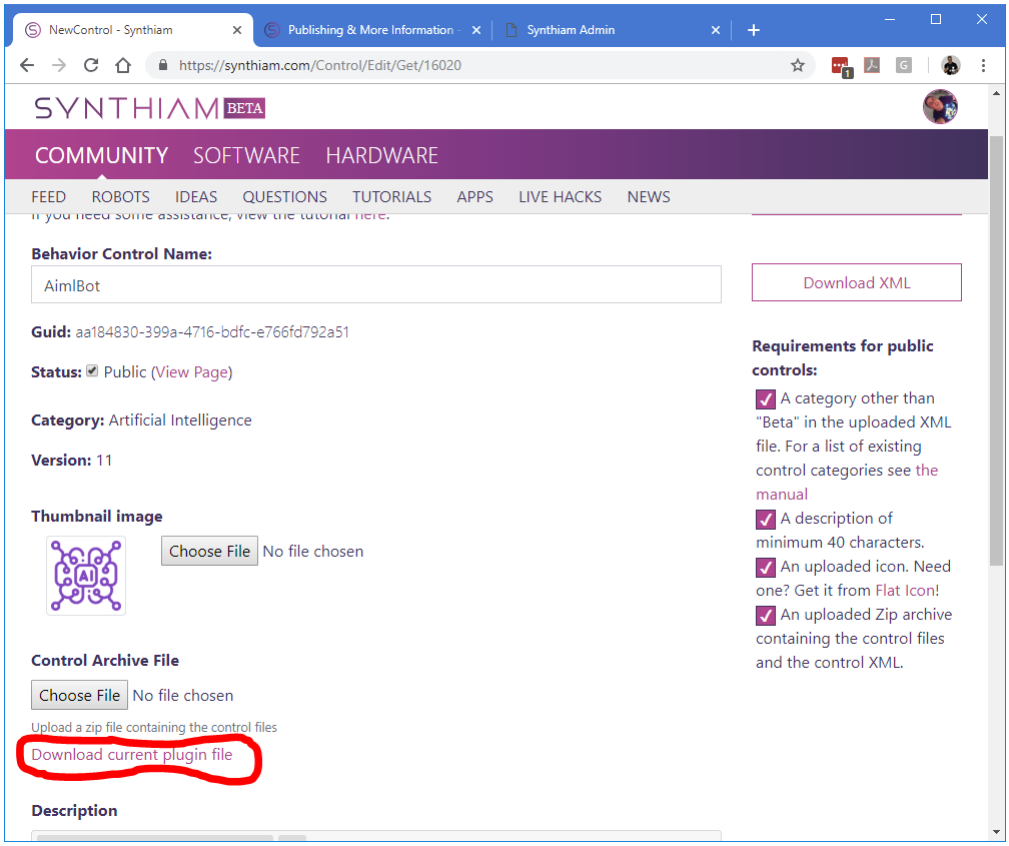
Additional Open-Source Code Examples Additional open-source code examples can be found by some developers. Some Synthiam skills are published OpenSource in our GitHub repository by clicking hereï»¿.

Adding Custom ARC Icons Your skill will use the default icon if there isn't another specified. To add an icon, include either a transparent PNG in your project output folder named **title.png**. This file will be used in the Add Skill window of ARC as an icon for your skill.

Changing ARC Category You may wish to change the category of your skill from BETA to an appropriate category when publishing to the public. The category is specified in the Plugin.xml file. The category is must match one of the categories from the ARC Add Control menu.

Share Unpublished Skill (User Testing) It is a smart idea to share your newly created skill with others before publishing to the public. You can copy the URL of the DOWNLOAD option on your Skill Definition page and send to others. The DOWNLOAD option will not display until a valid skill file has been uploaded to the skill Definition.

To share the install of the skill, simply copy the URL of the DOWNLOAD option or send the .PLUGIN to your test group.



Troubleshooting

This is a list of troubleshooting conversations with solutions from the community forum regarding creating robot skill plugins...

Blank Space In GUID Folder Name https://synthiam.com/Question/1465

Plugin.XML Not Copying to Output Folder https://synthiam.com/Question/1465

Plugin.XML file with invalid characters produces error https://synthiam.com/Question/4186

Here are some items to verify that you may have overlooked during the previous tutorial steps. Please check that these steps have been taken:

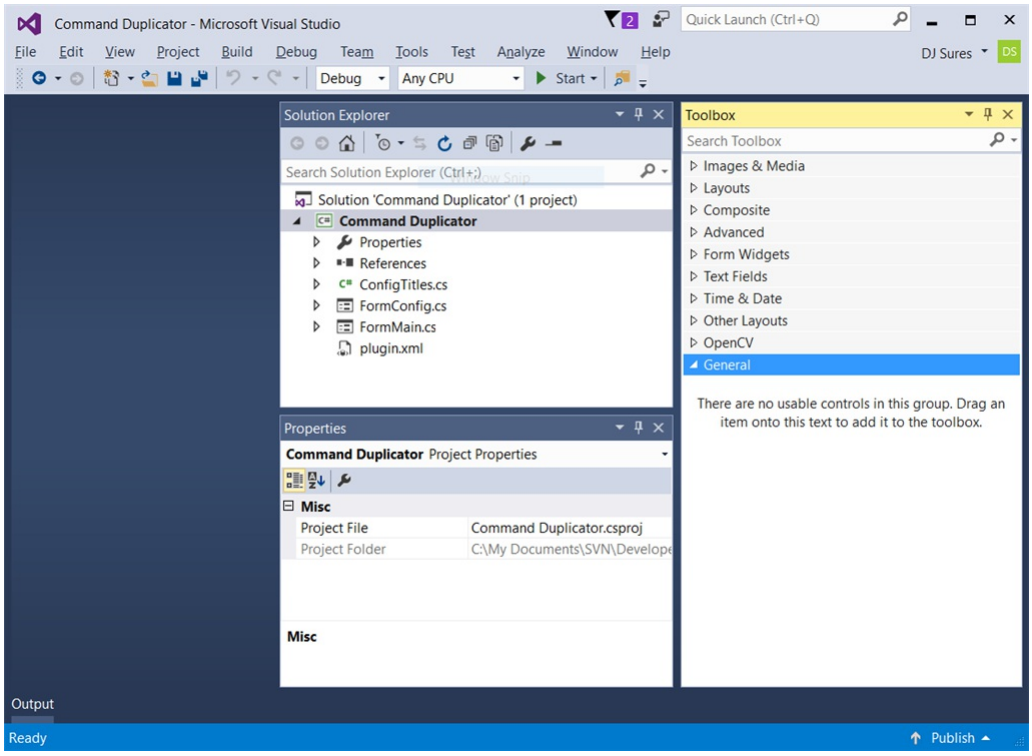
1. Have the ARC and EZ_B references been added and configured for "not to copy"?
2. Is the Output Folder of the skill specified to the correct location? (i.e. *c:\ProgramData\Arc\plugins*)
3. Is the plugin.xml configured to be copied to the output folder? Setting should be set for "Copy Always" in solution explorer properties.
4. Is the latest version of ARC installed?
5. Is the main form of your skill inheriting the correct PluginMaster class rather than Form class?
6. If you cannot execute the skill for debugging, is the "Debug With External Application" configured to use ARC.exe?
7. Is the correct DLL filename of your skill specified in the Plugin.xml file?
8. Is the correct GUID specified in the plugin.xml file?

To verify any of these questions, revisit the tutorial steps to ensure that you have completed the tutorial entirely.

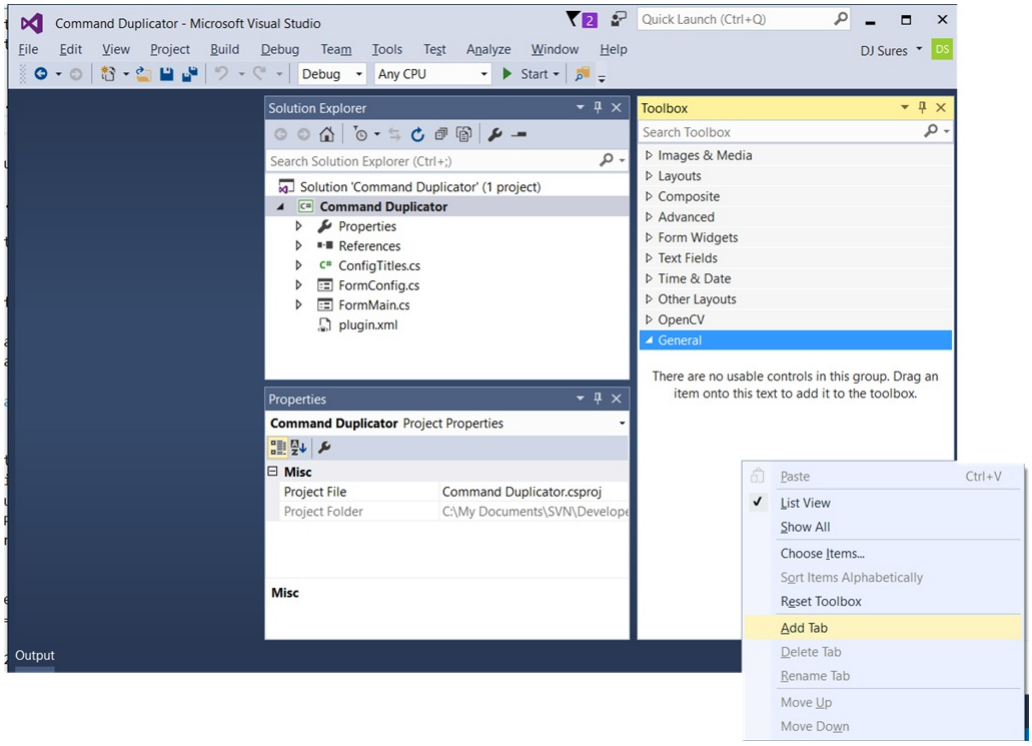
The ARC Skill Framework is very powerful because it is entirely open. That means you can access every component and resource within the ARC.exe application or EZ_B.DLL library. There are hundreds of custom .Net user UI components exposed in the ARC application to assist with your robot skill plugin. Including joysticks, buttons, camera canvases, and more.

Theming All skill components will be themed using the ARC standard theme renderer. This happens automatically, so the control colors, look and feel will be adjusted accordingly. There is an Example in another tutorial step regarding themes and the available commands to interact with the theme engine.

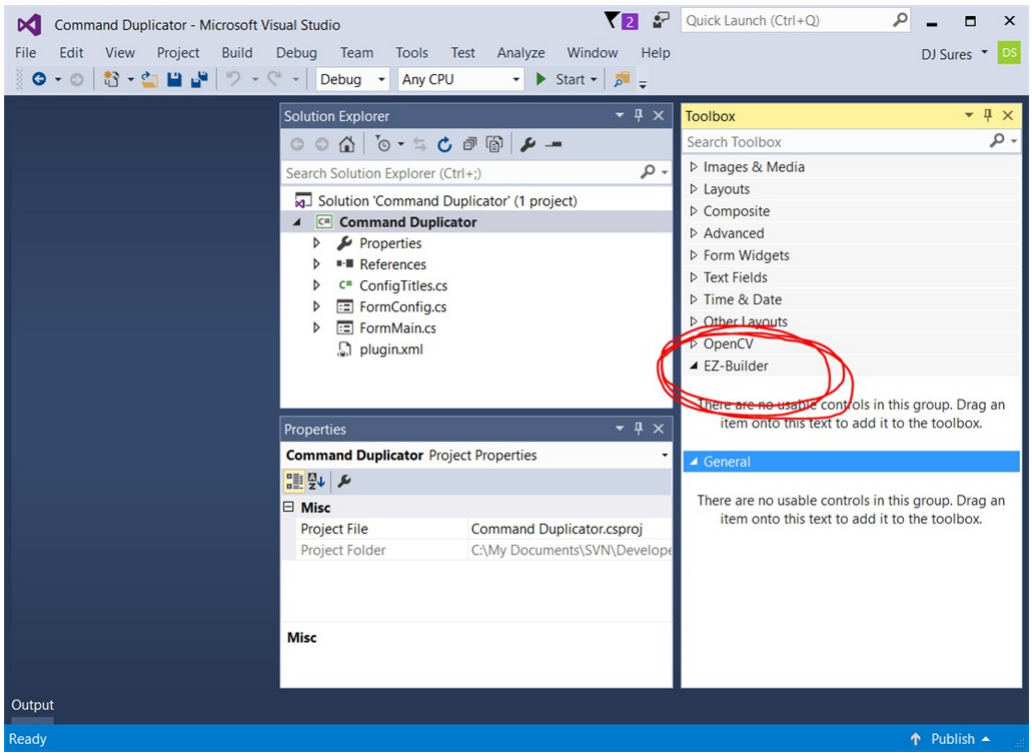
The Visual Studio IDE has a Tool Box, which is used for designer mode when customizing a form. The toolbox can have custom controls added to it. Below is a screenshot of a default toolbox.



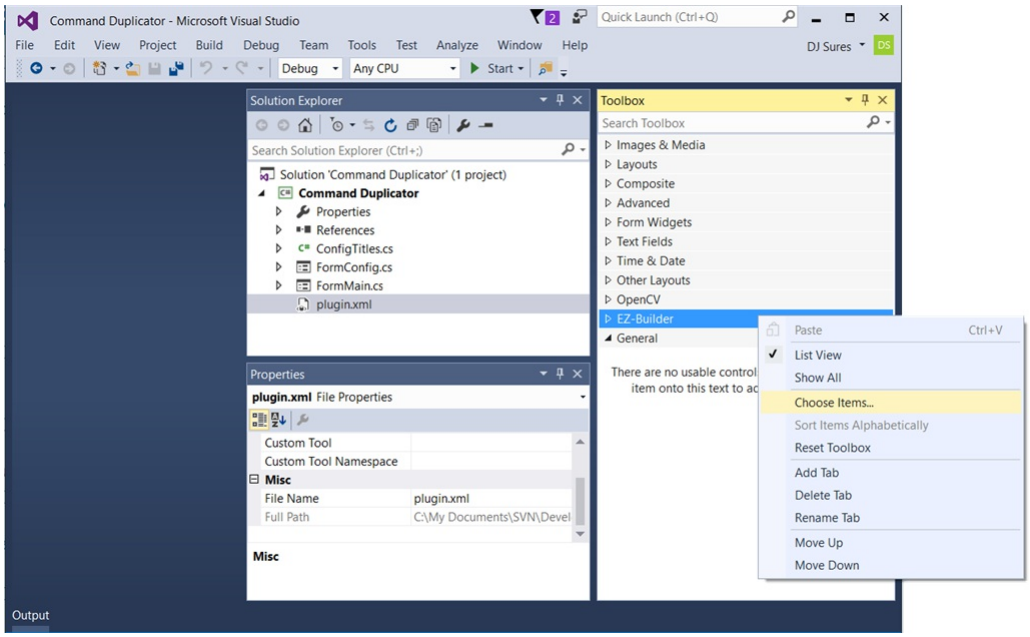
Create New Toolbox Tab To keep the new ARC component controls organized, we will create a new category in the toolbox. This is done by scrolling to the bottom of the toolbox list and RIGHT CLICK -> ADD TAB



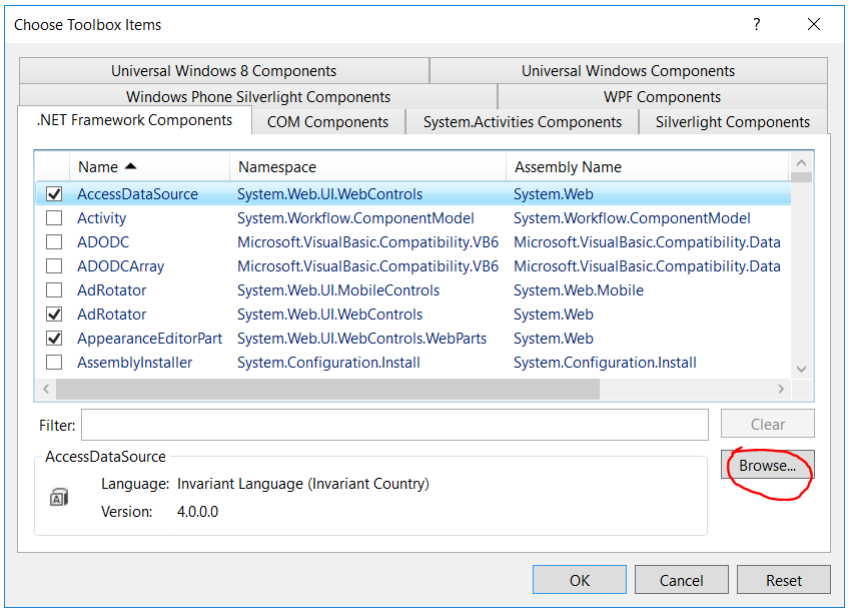
The new tab will be named ARC.



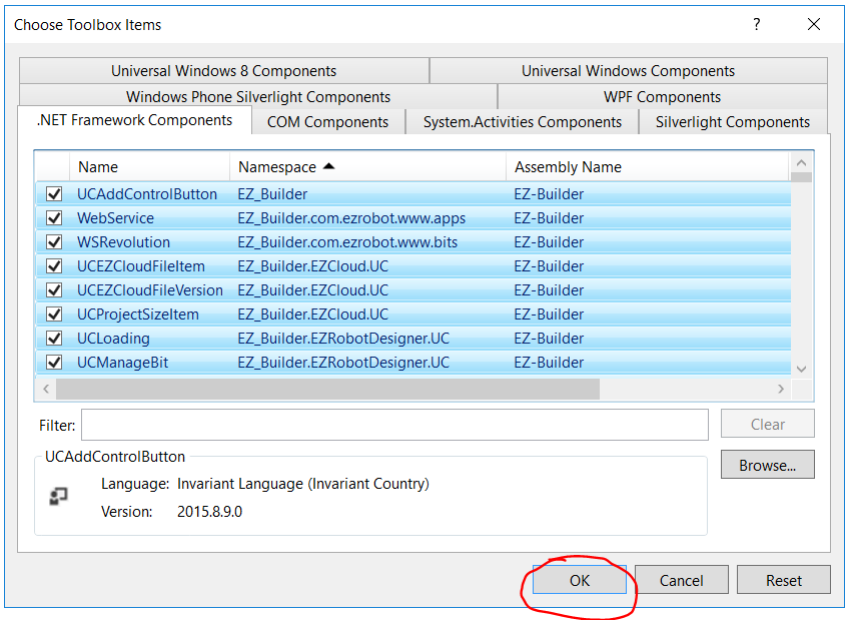
Add Controls To ARC Tab Now we will right click on the ARC tab and select CHOOSE ITEMS



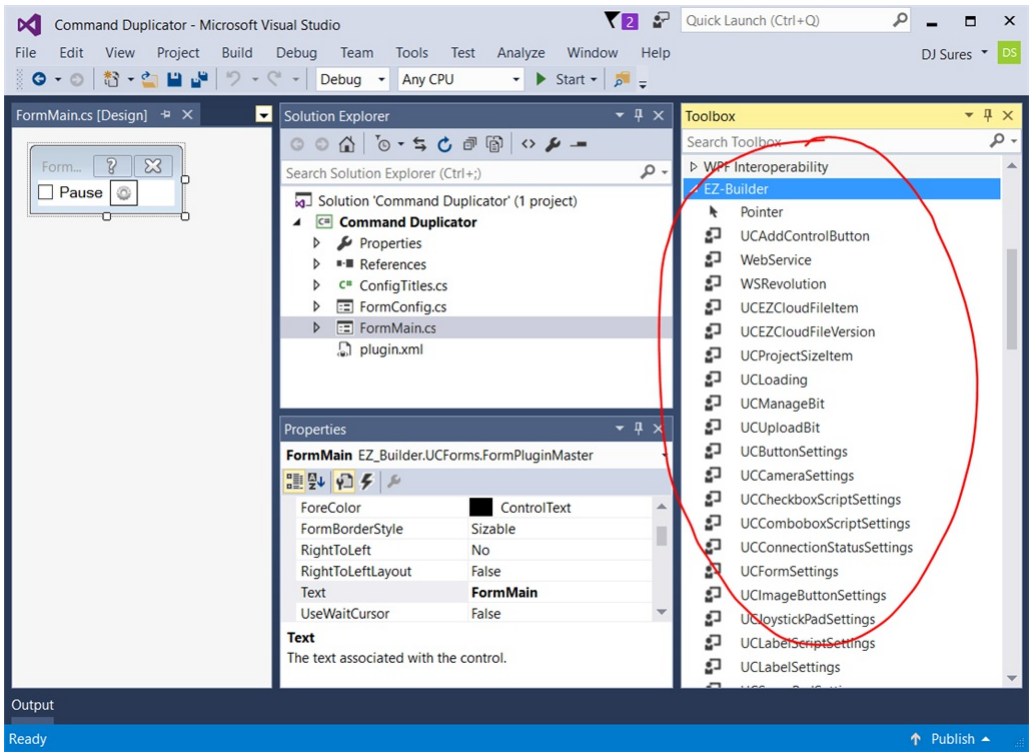
A Choose Items Dialog box will be displayed. It may take a few moments for the dialog to load while it scans and organizes the existing controls within the toolbox. Press the BROWSE button.



Now locate and select the ARC.exe application in C:\Program Files (x86)\Synthiam Inc\ARC. All controls of ARC will now be selected for the toolbox. Press OK



When editing/creating a Windows Form, the ARC tab in the toolbox will now have many new controls that you can begin using. Have fun!


Command Duplicator - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
DJ Sures
Debug
Any CPU
Start
FormMain.cs [Design]
Form...
Pause
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Command Duplicator' (1 project)
Command Duplicator
Properties
References
ConfigTitles.cs
FormConfig.cs
FormMain.cs
plugin.xml
Properties
FormMain EZ_Builder.UCForms.FormPluginMaster
ForeColor ControlText
FormBorderStyle Sizable
RightToLeft No
RightToLeftLayout False
Text FormMain
UseWaitCursor False
Text
The text associated with the control.
Toolbox
Search Toolbox
WPF Interoperability
EZ-Builder
Pointer
UCAddControlButton
WebService
WSRevolution
UCEZCloudFileItem
UCEZCloudFileVersion
UCProjectSizeItem
UCLoading
UCManageBit
UCUploadBit
UCButtonSettings
UCCameraSettings
UCCheckboxScriptSettings
UCComboboxScriptSettings
UCConnectionStatusSettings
UCFormSettings
UCImageButtonSettings
UCJoystickPadSettings
UCLabelScriptSettings
UCLabelSettings
Output
Ready
Publish

Example: EZB Manager

The ARC.EZBManager is a helper class that makes accessing I/O very easy, by doing the complicated work for you. Within the EZBManager are methods to read adc, read digital, set digital, set servo positions, etc..

The ARC.EZBManager also has an array of EZB objects. Each EZB object is a physical EZB. This is an array because the userâ€™s project can have up to 128 EZ-Bâ€™s defined. Itâ€™s always safe to assume that the first (index zero) EZ-B is default, which is why EZBManager.EZBs[0] is commonly used for raw commands that are not using the EZBManager helper methods.

It is highly recommended to use the ARC.EZBManager instead of raw commands directly to the EZB. Here are some examples of using the ARC methods, to demonstrate how much more convenient they are for moving servos, specifically...

``` ARC.EZBManager.SetServoPosition(_cf.SERVOS[ConfigurationDictionary._VERTICAL_SERVOS], 20);

ARC.EZBManager.SetServoIncrement(_cf.SERVOS[ConfigurationDictionary._HORIZONTAL_SERVOS], -1); ```
```

Example: Saving/Loading Configuration

In all plugins, there is a _cf variable which is provided by the inherited class which defines the Form. The _cf (configuration file) contains the data saved with your project. The _cf.STORAGE[key] is where your plugin configuration is kept. However, due to EZ-Builder providing the UCServoSelection user control, there is also a _cf.SERVOS[key] to store servo configuration.

The UCServoSelection user control is the common displayed control across all configuration screens in EZ-Builder. This user control allows configuration of servo ports, min and max distances. As well as multi servo support, etc.. So, if your plugin is to use the UCServoSelection, we highly recommend doing so. To move servos with UCServoSelection configuration data, the example source code in the above mentioned project should be reviewed. In short, it's easy to pass UCServoSelection.Config() data to the ARC.EZBManager.SetServoPosition(), which will take care of the magic - including relationship scaling between multiple servos, min, max and inverted options from UCServoSelection.

Configuration settings must be initialized with default values in the SetConfiguration override. This will ensure that when your plugin is loaded, there is some default configuration data assigned to the keys. Because the data is initialized using the _cf.STORAGE.AddIfNotExist(), the default data will only be added if the key doesn't exist. If the key does exist, following the user loading a project with saved configuration data, the default configuration will not be applied to the key. As demonstrated... ``` public override void SetConfiguration(EZ_Builder.Config.Sub.PluginV1 cf) {

```
  cf.SERVOS.AddIfNotExist(ConfigurationDictionary._HORIZONTAL_SERVOS, new EZ_Builder.Config.Sub.ServoDescriptor[] { });
  cf.SERVOS.AddIfNotExist(ConfigurationDictionary._VERTICAL_SERVOS, new EZ_Builder.Config.Sub.ServoDescriptor[] { });

  cf.STORAGE.AddIfNotExist(ConfigurationDictionary._HORIZONTAL_DEGREES, 0.14m);
  cf.STORAGE.AddIfNotExist(ConfigurationDictionary._VERTICAL_DEGREES, 0.14m);
  cf.STORAGE.AddIfNotExist(ConfigurationDictionary._EDGE_ENABLED, true);
  cf.STORAGE.AddIfNotExist(ConfigurationDictionary._EDGE_SIZE, 10);

  base.SetConfiguration(cf);
}
```

```

The base.SetConfiguration in the above example is necessary to set the configuration with the inherited base. This ensures the initialized configuration is applied.

Also in the above example, the ConfigurationDictionary is a class which contains read-only static strings to reference the resource keys. We use these strings so the configuration data can be referenced without worrying about spelling mistakes where ever the data is to be accessed. Here is a look at the above example ConfigurationDictionary...

``` public class ConfigurationDictionary {

```
public static readonly string _HORIZONTAL_SERVOS = "horizontal servos";
public static readonly string _VERTICAL_SERVOS = "vertical servos";

public static readonly string _HORIZONTAL_DEGREES = "horizontal degrees";
public static readonly string _VERTICAL_DEGREES = "vertical degrees";

public static readonly string _EDGE_SIZE = "edge size";
public static readonly string _EDGE_ENABLED = "edge enabled";
```

} ```

Now to use the configuration data from the _cf in your project, simply cast the object from the _cf.STORAGE to the correct type. For _cf.SERVOS, pass the key result directly into the ARC.EZBManager helper methods. Like so...

``` if (Convert.ToBoolean(_cf.STORAGE[ConfigurationDictionary._EDGE_ENABLED])) {

```
    int edgeSize = Convert.ToInt16(_cf.STORAGE[ConfigurationDictionary._EDGE_SIZE]);
    var servosX = _cf.SERVOS[ConfigurationDictionary._HORIZONTAL_SERVOS];
    var servosY = _cf.SERVOS[ConfigurationDictionary._VERTICAL_SERVOS];

    if (_mousePoint.X _cameraControl.Camera.CaptureWidth - 10)
      ARC.EZBManager.SetServoIncrement(servosX, 1);

    if (_mousePoint.Y _cameraControl.Camera.CaptureHeight - 10)
      ARC.EZBManager.SetServoIncrement(servosY, 1);
  }
```

```

Any data in the _cf.STORAGE and _cf.SERVOS will be automatically saved with the users project. And when that project is loaded again in the future, the _cf data will be reloaded as well. This allows your plugin to save the custom configuration applied by the user.

Example: Moving Robot

If your plugin is to make a robot move, it doesn't need to know anything about the robot.

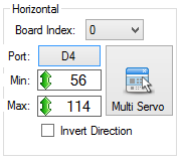
ARC uses a concept of â€œMovement Panelsâ€. Users can only add one movement panel per project, and this is how their robot moves. When a movement panel is added to a project, it binds itself to a common class that is exposed to the plugin system. This allows your program to easily tell the robot to move forward, left, right, stop, etc. regardless of the movement panel the user has added. This means if the robot is a humanoid, it will walk forward using the project's own method of walking. If the project is a hexapod, it will do the same. If the robot is a drone, it will also do the same. So for your plugin, it doesnâ€™t matter what kind of robot it is â€“ you simply tell it to move a direction. And hereâ€™s a few examples...

``` ARC.EZBManager.MovementManager.GoForward(); ARC.EZBManager.MovementManager.GoReverse();
ARC.EZBManager.MovementManager.GoRight(); ARC.EZBManager.MovementManager.GoRollLeft();
ARC.EZBManager.MovementManager.GoUp();

Etc... ```

Within that Movement class, there are methods for moving in all directions. Up, Down, Left, Right, Stop, etc.. Not all movement panels will support Up, Down, RollLeft, RollRight as those are generally reserved for used with flying drone robots - but you get the point

**Note:** It is important to note that all native ARC Movement Panels use the movement class, as it is standard practice to put the movement servos/hbridges on the first EZB when designing a robot.
```

Example: Move Servos

If you wish to move a servo, it can be done one of two ways. You can specify the servo to move directly, or take advantage of ARC built in servo helpers.

The servo helpers are what you find throughout the ARC interface, when prompting users to select a servo in configuration screens. The user can always select one or more servos that have a relationship to each other, including inverting. It also allows the user to specify a MIN and MAX for each servo.



The Servo Helper in ARC makes moving servos a breeze - and that will make your plugin interact with servos much easier, without having to perform all the relationship math and limits yourself.

Servo Movements (Raw) To move a servo with raw commands, not using the servo helper, here is path to the class:

â€¢ ARC.EZBManager.EZBs[0].Servo

It is non recommended to set servo positions using this method, unless they are non user-configurable servos. If your plugin expects users to configure the servos, use the UCServoSelection object and the helper methods below.

Servo Movements (UCServoSelection) The ARC UI has the ability for users to specify multiple servos and have the positions relative to each other with invert options, etcâ€¦ This can be viewed in the sourcecode for the Click Servo plugin here: https://synthiam.com/Software/Manual/16147

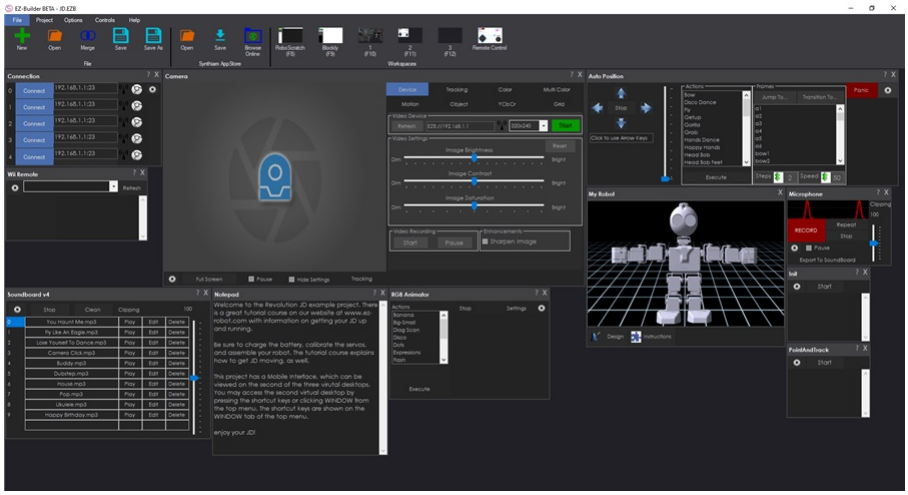
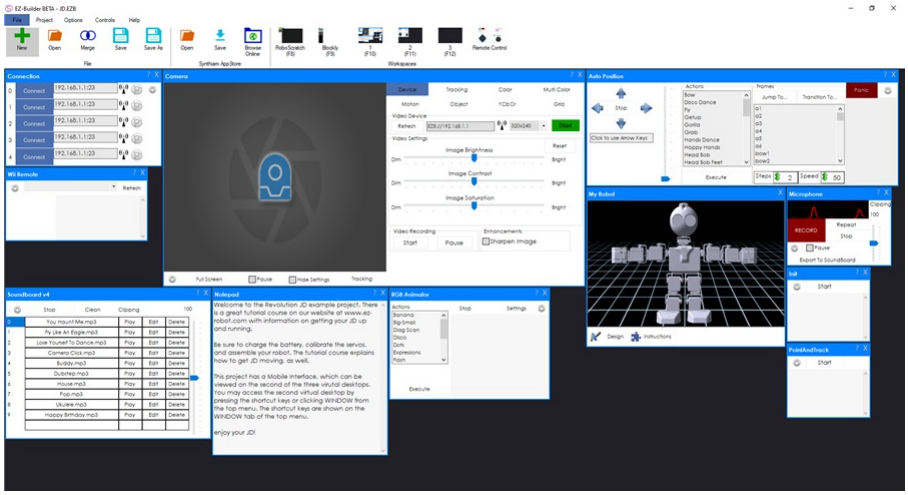
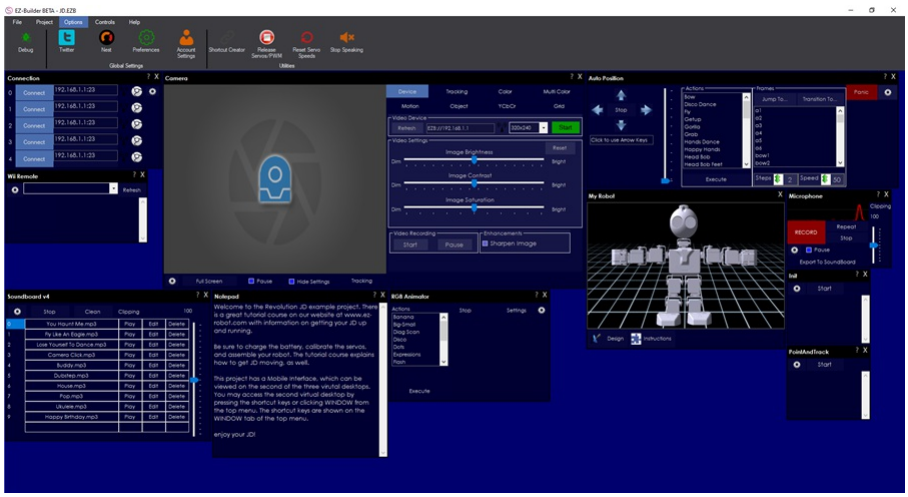
You can install that plugin to see how it works, and watch the video on how to set it up. The example in that source code demonstrates how to use the UCServoSelection user control for users to specify multiple servos, if you choose to go this route. You will notice that the array collection of servos is passed to helper commands in the ARC.EZBManager directly.

By using the UCServoSelection and ARC.EZBManager helper methods, the servo configuration can be easily stored in the project file and passed to helper methods for executing. For example, to extract and store the user's servo configuration from a UCServoSelection into the project file can be like so...

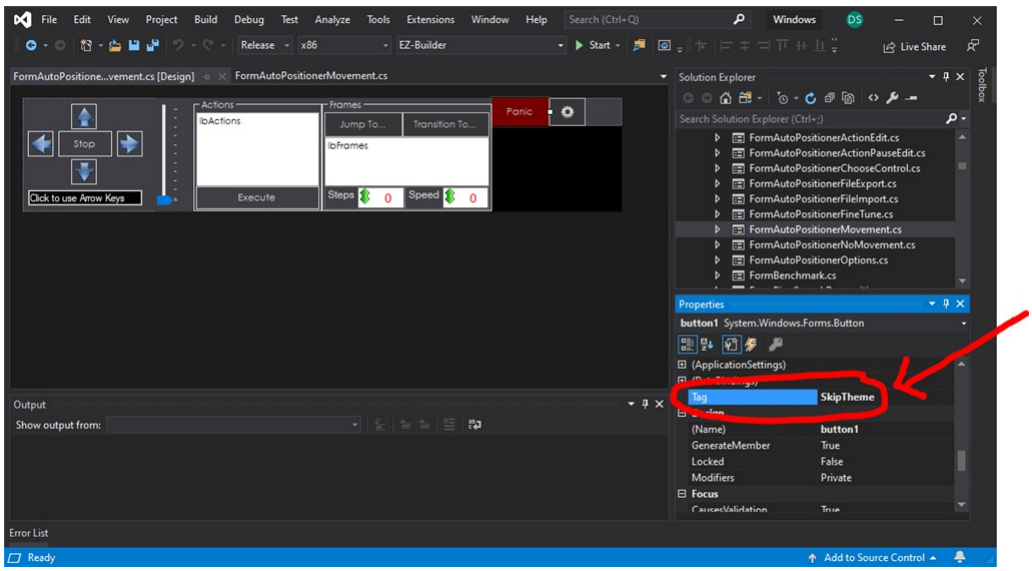
``` _cf.SERVOS.AddOrUpdate(ConfigurationDictionary._HORIZONTAL_SERVOS, ucServoSelectionX.Config);

_cf.SERVOS.AddOrUpdate(ConfigurationDictionary._VERTICAL_SERVOS, ucServoSelectionY.Config); ```

Now to move the servos from the configuration _cf.SERVOS file, simply do the following...

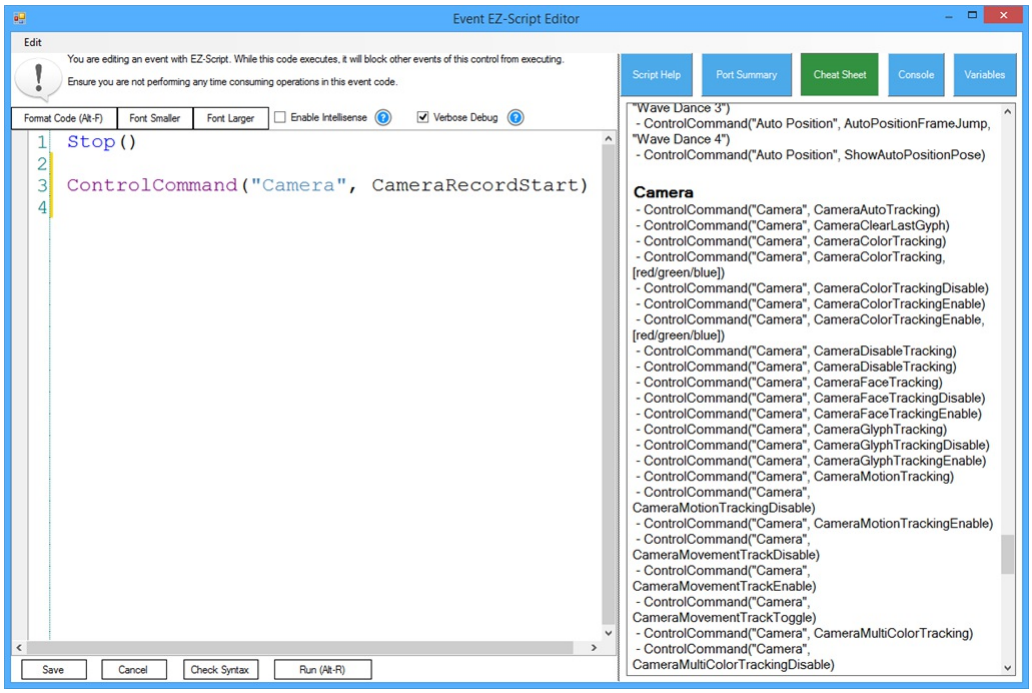
``` ARC.EZBManager.SetServoIncrement(_cf.SERVOS[ConfigurationDictionary._HORIZONTAL_SERVOS], -1);

ARC.EZBManager.SetServoIncrement(_cf.SERVOS[ConfigurationDictionary._VERTICAL_SERVOS], 1); ```

# Example: Theme Renderer

**Overview** When a behavior control is added to an EZ-Builder workspace by a user, the control form UI will be manipulated to provide a standard look and feel. Research has demonstrated the benefits of providing users with an unified graphical experience that promotes creativity within EZ-Builder. This is done by relieving cognitive load of the user by giving them less to *think* about during their robot programming sessions.

**Some Theme Examples** Users configure theme colors for their EZ-Builder instance on a per user basis. This configuration is stored in the current logged-in user's registry. The color theme can be customized in the top EZ-Builder menu *Options -> Preferences -> Window Theme*.







**When Is A Control Form Themed?** The theme engine is called against forms when they're added to the project automatically by the ARC workspace manager. This is the same manager that allows changing desktops, smart arranging windows, loading configurations, etc.. As far as when the theme renderer is called in code, it's after the form's constructor and before the OnLoad() event.

``` YourPlugin.Constructor() | V Theme Renderer | V YourPlugin.OnLoad() ``` **Skip Themeing of Controls** There's two ways to have a control skip the theme process. This means the specified controls, and respective child controls will be skipped when the theme is applied.

1. ThemeRenderer - The *FormPluginMaster* is the base form that your plugin must inherit. Within this *FormPluginMaster* is a *ThemeRenderer* object. You can access the ThemeRenderer.ControlsToSkipThemingï»¿ in the plugin form's constructor following
```

InitializeComponent()Â Â For example, if you had a btnSave that wished to not be themed...

``` public MyPlugin() {

// Define and initialize components on plugin form InitializeComponent();

// Skip the following components from the theme renderer on form ThemeRenderer.ControlsToSkipTheming.Add(btnSave); ï»¿} ``` 2) Tag: SkipThemeï»¿ - In the properties of a control on your plugin form, you can specify *SkipTheme* as the *Tag* value. This will instruct the ThemeRenderer to skip the control and all child controls.


```

Example: ControlCommand() Binding

Overview The ControlCommand() is a scripting function which enables users to send commands and parameters to supporting controls from another control. It is how controls communicate. Each control broadcasts a list of commands it supports. These commands are displayed in the Cheat Sheet while users are editing scripts (JavasScript or EZ-Script). When a ControlCommand() is executed that has a destination name matching your plugin, an event is triggered in which you will respond.



An Example An example of a popular ControlCommand() is starting the video feed in the camera control. Users will add Script Control to their program which instructs the Camera Control to begin streaming video from the specified video source.

```

# JavaScript example to start the video feed on a camera control

controlCommand("Camera Control", "CameraStart"); ```

You may notice that some ControlCommand() will accept optional parameters. The CameraStart also has an optional parameter, which is the device name.

```

JavaScript example to start the video feed on a camera control specifying device name

controlCommand("Camera Control", "CameraStart", "EZB://192.168.1.1:24"); ```

Bind To ControlCommand() Now that the user is aware of the supported options available in your Cheat Sheet, we will bind to the script engine for any calls directed to your control. This is done through an override method which will be raised in the event that a ControlCommand() matches your control.

Some facts to note in this example code...

1. Comparison is case insensitive. We have no idea what case the text will be entered by the user.
2. If no commands match your syntax, the Base() method will notify the script engine.
3. If expected parameters are missing or incorrect, you may throw an exception which will be caught by the parent script engine.
4. To avoid cross-threading exceptions, there is a fancy helper class ARC.Invokers which contains methods to manipulate user controls from different threads. The SendCommand() event will *always* be called from a background thread. This is because the script engine will never execute threads on a GUI thread.

``` public override void SendCommand(string windowCommand, params string[] values) {

```
  if (windowCommand.Equals("PauseOn", StringComparison.InvariantCultureIgnoreCase)) {

    ARC.Invokers.SetChecked(checkbox1, true);

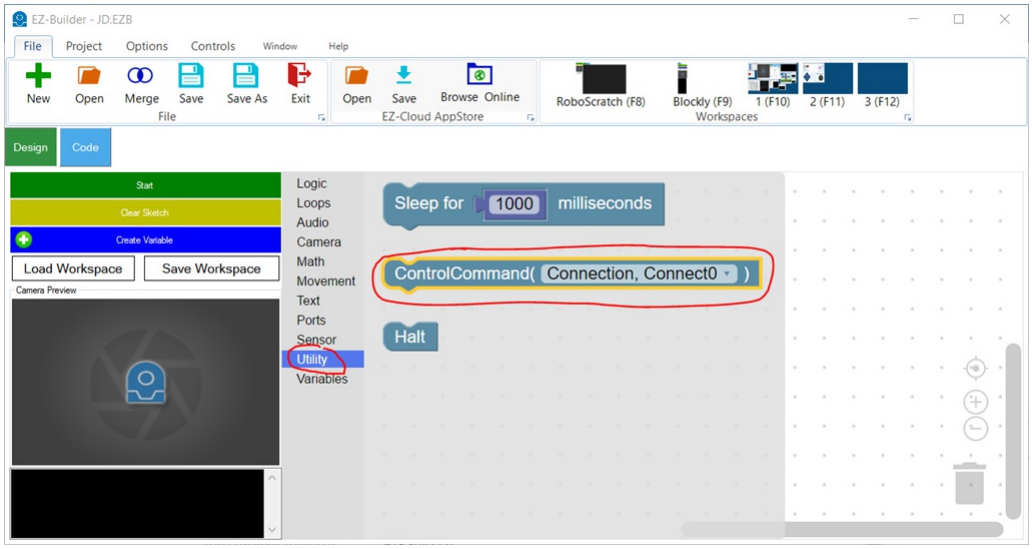
    if (values.Length == 1)
      ARC.Invokers.SetText(checkbox1, values[0]);
    else if (values.Length > 1)
      throw new Exception(string.Format("Only 0 or 1 parameters expected. You passed {0}", values.Length));
```
```

```
  } else if (windowCommand.Equals("PauseOff", StringComparison.InvariantCultureIgnoreCase)) {

    ARC.Invokers.SetChecked(checkbox1, false);

    if (values.Length == 1)
      ARC.Invokers.SetText(checkbox1, values[0]);
    else if (values.Length > 1)
      throw new Exception(string.Format("Only 0 or 1 parameters expected. You passed {0}", values.Length));

  } else {

    base.SendCommand(windowCommand, values);
  }
}

```

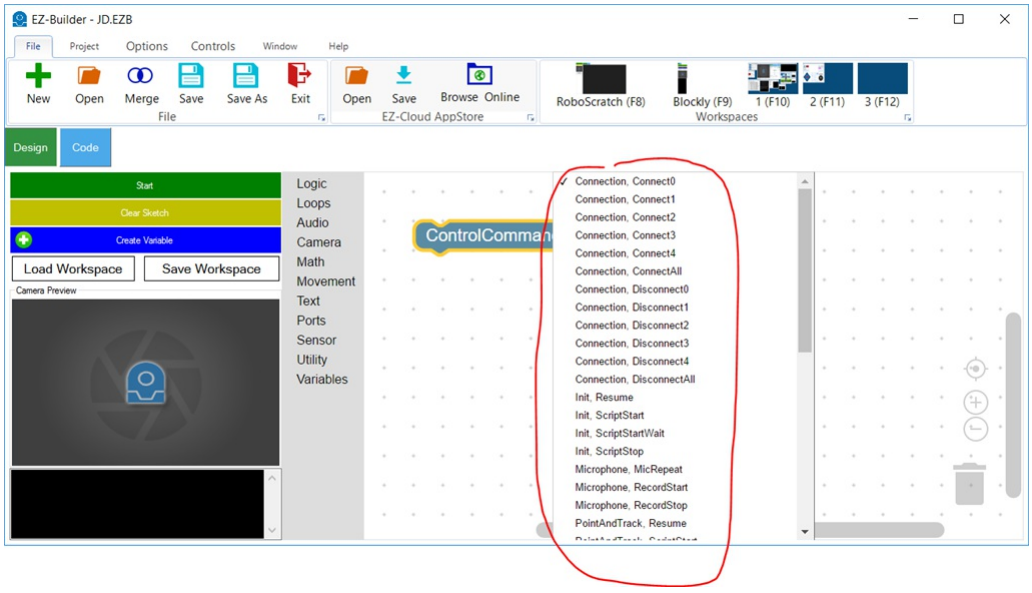
Conclusion By using the ControlCommand(), users can send commands to your plugin or configure settings, all from scripts. This gives your plugin the ability to be better customized for the users needs programmatically.

Blockly ControlCommand() are usable in Blockly UI, with one exception. Because the Blockly UI does not contain the ability for user defined parameters of the ControlCommand() feature, they are limited to commands with no user parameters. This means that a ControlCommand() with parameters will not display in the blockly UI.

The ControlCommand() for blockly is found in the Utility category.



Viewing the available ControlCommand()'s within blockly, you will see that commands accepting user parameters are not displayed..



To further the example, here are two control commands in which one will be displayed, and one will not be ``` // This will be displayed in blockly controlCommand("My Control", "SetColorRed"); controlCommand("My Control", "SetColorGreen");

// These will not be displayed in blockly because it accepts a user parameter controlCommand("My Control", "SetColor", "Red"); controlCommand("My Control", "SetColor", "Green"); ```

Example: Camera Control

Your plugin can access the camera by attaching itself to an existing camera control. Attaching to an existing camera control is a friendly way of sharing the camera video stream with other plugins and maintaining the camera control's features.

This is demonstrating one method of finding a control within the workspace. There are a number of methods that make this easy. Take a look at the *Example: Finding Other Behavior Control* section of this tutorial for a more detailed explanation.

In this example, we will search for an existing camera control and attach to frame event.

Step 1 - Declare a global camera control variable

Your plugin will need to search the project to find a camera control. When it finds the camera control, it will need to keep a reference to it for future actions. Specifically, your plugin will need to keep the camera control reference so it can detach from the video frame event during dispose or close.

``` namespace Camera_Example {

public partial class FormMain : EZ_Builder.UCForms.FormPluginMaster {

```
// Global variable within your plugin class
ARC.UCForms.FormCameraDevice _cameraControl;

public FormMain() {

  InitializeComponent();
}
```

} } ```

**Step 2 - Create an attach() method with Frame Event**

This method will search the project for an existing camera control, add a reference to it and attach a method to the frame event.

``` void attach() {

// detach in case there is already an attachment to an existing camera control detach();

// get all camera controls in the project Control [] cameras = ARC.EZBManager.FormMain.GetControlByType(typeof(ARC.UCForms.FormCameraDevice));

// if there are no camera controls, inform the user if (cameras.Length == 0) {

```
MessageBox.Show("There are no camera controls in this project.");

return;
```

}

// get a reference to the first camera control we find, in the case there are many _cameraControl = (ARC.UCForms.FormCameraDevice)cameras[0];

// attach to the New Frame event _cameraControl.Camera.OnNewFrame += Camera_OnNewFrame;

ARC.EZBManager.Log("Attached to: {0}", _cameraControl.Text); }

// We will add code to this method later in this example void Camera_OnNewFrame() {

} ```

Step 3 - Create a detach() method to detach from the camera control

Now that we have created an attach() method, there will need to be a method to detach as well. The detach is necessary specifically for your plugin's Closing event. Be sure to add the detach to the Closing event as in this example.

``` private void FormMain_FormClosing(object sender, FormClosingEventArgs e) {

// detach the event handler from this plugin when it closes detach(); }

void detach() {

// only perform detach if there is a reference to a camera control if (_cameraControl != null) {

```
ARC.EZBManager.Log("Detaching from {0}", _cameraControl.Text);

// Detach the from the camera event handler
_cameraControl.Camera.OnNewFrame -= Camera_OnNewFrame;

// Set the camera control reference to null
_cameraControl = null;
```

} } ```

**Step 4 - Create a button to Attach()**

Your plugin will need some way to let the user attach to the camera control. You could do this when the plugin is initiated, but you may wish to give the user the ability to attach/detach on their own. Just in case they do not wish for it to be active right away. Create a button on your plugin which will have this code in the OnClick event. This code will attach() if there is no reference to a camera control, and detach if there is. The code will work as a toggle between the two states.
```

``` private void btnAttach_Click(object sender, EventArgs e) {

if (_cameraControl == null) attach(); else detach(); } ```

**Step 5 - Create ControlCommand() attach/detach**

Be friendly to your users by giving them the ability to attach and detach your plugin via ez-script ControlCommand() syntax. These two override methods will do just that...

``` // Override is called when ControlCommand() is sent to this control // If an applicable command is passed, execute it. Otherwise execute the base which throws an exception to the user public override void SendCommand(string windowCommand, params string[] values) {

if (windowCommand.Equals("attach", StringComparison.InvariantCultureIgnoreCase)) attach(); else if (windowCommand.Equals("detach", StringComparison.InvariantCultureIgnoreCase)) detach(); else base.SendCommand(windowCommand, values); }

// Return a list of available ControlCommands() to the ï»¿ARC UI // This list is presented to the user when they are editing EZ-Script and viewing the Cheat Sheet public override object[] GetSupportedControlCommands() {

List cmds = new List();

cmds.Add("Attach"); cmds.Add("Detach");

return cmds.ToArray(); } ```

Step 6 - Do something in the video frame event

Now that you have successfully created attach and detach methods to the camera control, let's do something with the video frame event. The video frame event was created earlier in this project, and was empty. Now let's simply add some code to get the image and draw on it.

Add a reference to the *aforge.dll* in the ARC program folder. This is the same folder which you added the ARC.exe and EZ_B.dll during the plugin setup. The aforge.dll library is a fantastic open-source project with direct memory access to image buffer for manipulation, which is faster than using GDI (System.Drawing).

In this example frame event code, we will draw a solid square around the object which was detected during tracking. For testing, you will need to enable a tracking type.

``` void Camera_OnNewFrame() {

// Exit this method if there is no reference to a camera control // This may only happen during a dipose/detach if the frame is already executing if (_cameraControl == null) return;

// If no object is detected, display a message to the user in the camera image if (ARC.Scripting.VariableManager.GetVariable("$CameraIsTracking") == "0") {

```
using (Graphics g = Graphics.FromImage(_cameraControl.Camera.GetOutputBitmap.ToManagedImage(false)))
  g.DrawString("No object detected", SystemFonts.CaptionFont, Brushes.Red, 0, 0);

return;

}
```

// If we got this far, an object must be detected // Get the camera control variables for the detected object location and size int objectX = Convert.ToInt32(ARC.Scripting.VariableManager.GetVariable("$CameraObjectX")); int objectY = Convert.ToInt32(ARC.Scripting.VariableManager.GetVariable("$CameraObjectY")); int objectCenterX = Convert.ToInt32(ARC.Scripting.VariableManager.GetVariable("$CameraObjectCenterX")); int objectCenterY = Convert.ToInt32(ARC.Scripting.VariableManager.GetVariable("$CameraObjectCenterY")); int objectWidth = Convert.ToInt32(ARC.Scripting.VariableManager.GetVariable("$CameraObjectWidth")); int objectHeight = Convert.ToInt32(ARC.Scripting.VariableManager.GetVariable("$CameraObjectHeight"));

// Draw a Crosshair using the aforge library fast draw functions AForge.Imaging.Drawing.Line(_cameraControl.Camera.GetOutputBitmap, new AForge.IntPoint(objectCenterX - 8, objectCenterY), new AForge.IntPoint(objectCenterX + 8, objectCenterY), Color.FromArgb(200, 240, 100, 200)); AForge.Imaging.Drawing.Line(_cameraControl.Camera.GetOutputBitmap, new AForge.IntPoint(objectCenterX, objectCenterY - 8), new AForge.IntPoint(objectCenterX, objectCenterY + 8), Color.FromArgb(200, 240, 100, 200));

// draw a filled rectangle with opacity AForge.Imaging.Drawing.FillRectangle(_cameraControl.Camera.GetOutputBitmap, new Rectangle(objectX, objectY, objectWidth, objectHeight), Color.FromArgb(100, 50, 50, 250)); } ```

**You're Done!**

You have successfully created methods required to attach/detach from existing camera controls and perform drawing on the frame image!
```

Example: Camera Custom Tracking Type

As you've seen in the previous tutorial step about detecting and attaching to the camera, there are a bunch of events that you can use. One of the events allows you to create a custom tracking type as a plugin, which is real cool!

Uses for creating a custom tracking type is if you want to experiment with OpenCV or any other vision libraries. Because ARC leverages .Net, we recommend the x86 nuget install of EMGUCV (https://github.com/emgucv/emgucv). Installing from NUGET is the easiest and most convenient.

The camera events that we'll use for creating a custom tracking type are...

``` // assign an event that raises when the camera wants to initialize tracking types _camera.Camera.OnInitCustomTracking += Camera_OnInitCustomTracking;

```
    // assign an event that raises with a new frame that you can use for tracking
    _camera.OnCustomDetection += Camera_OnCustomDetection;
```

```

Once inside the OnCustomDetection() event, you have access to a bunch of different bitmaps throughout the flow of the detection process. They are...

``` // ********** // From the EZ_B.Camera class // **********

```
///
/// This is a temporary bitmap that we can use to draw on but is lost per tracking type
///
public volatile Bitmap _WorkerBitmap;

///
/// This is the resized original bitmap that is never drawn on. Each tracking type uses this as the main source image f
///
public volatile AForge.Imaging.UnmanagedImage _OriginalBitmap; // resized image that we process

///
/// Image that is outputted to the display. We draw on this bitmap with the tracking details
///
public volatile AForge.Imaging.UnmanagedImage _OutputBitmap;

///
/// Raw image unsized directly from the input device
///
public volatile AForge.Imaging.UnmanagedImage _RawUnsizedBitmap;

///
/// Last image for the GetCurrentImage
///
public volatile AForge.Imaging.UnmanagedImage _RawUnsizedLastBitmap;
```

```

Understanding the images available, the ones we care about for creating a tracking type of our own are...

``` /// /// This is the resized original bitmap that is never drawn on. Each tracking type uses this as the main source image for tracking, and then draws on the _OutputBitmap for tracking details /// public volatile AForge.Imaging.UnmanagedImage _OriginalBitmap; // resized image that we process

```
///
/// Image that is outputted to the display. We draw on this bitmap with the tracking details
///
public volatile AForge.Imaging.UnmanagedImage _OutputBitmap;
```

```

This is because we can use the _OriginalBitmap for our detection, and then draw on the _OutputBitmap where our detection was.

Example This is an example that fakes detection by drawing a rectangle on the _OutputBitmap that bounces around the screen. It moves with every frame in the CustomDetection event.

``` // faking an object being tracked int _xPos = 0; int _yPos = 0; bool _xDir = true; bool _yDir = true;

```
private EZ_B.ObjectLocation[] Camera_OnCustomDetection(EZ_Builder.UCForms.FormCameraDevice sender) {

  if (_isClosing)
    return new ObjectLocation[] { };

  if (!_camera.Camera.IsActive)
    return new ObjectLocation[] { };

  List objectLocations = new List();

  try {

    // This is demonstrating how you can return if an object has been detected and draw where it is
    // The camera control will start tracking when more than one ObjectLocation is returned
    // We're just putting fake bouncing rectable of a detected rect which will be displayed as a tracked object on the

    if (_xDir)
      _xPos += 10;
    else
```
```

```
      _xPos -= 10;

    if (_yDir)
      _yPos += 10;
    else
      _yPos -= 10;

    var r = new Rectangle(_xPos, _yPos, 50, 50);

    if (r.Right > _camera.Camera._OutputBitmap.Width)
      _xDir = false;
    else if (r.Left  _camera.Camera._OutputBitmap.Height)
      _yDir = false;
    else if (r.Top <= 0)
      _yDir = true;

    var objectLocation = new ObjectLocation(ObjectLocation.TrackingTypeEnum.Custom);
    objectLocation.Rect = r;
    objectLocation.HorizontalLocation = _camera.Camera.GetHorizontalLocation(objectLocation.CenterX);
    objectLocation.VerticalLocation = _camera.Camera.GetVerticalLocation(objectLocation.CenterY);
    objectLocations.Add(objectLocation);

    AForge.Imaging.Drawing.Rectangle(_camera.Camera._OutputBitmap, r, Color.MediumSeaGreen);
  } catch (Exception ex) {

    EZ_Builder.EZBManager.Log(ex);
  }

  return objectLocations.ToArray();
}

```

Example: Global Script Variables

The ARC variable manager stores and retrieves global variables that are available within ARC plugins and controls. This allows you to Set and Get variables that other controls may be configuring or using.

Data Types The script variable stores dynamic data types. This means that there is no declaration between numeric or string values. For example in JavaScript you would type... ``` setVar("$a", 3); setVar("$b", 3.123); setVar("$c", "This is a string"); setVar("$d", 0x55); ```

And in C# Plugin you would type... ``` ARC.Scripting.VariableManager.SetVariable("$a", 3); ARC.Scripting.VariableManager.SetVariable("$b", 3.123); ARC.Scripting.VariableManager.SetVariable("$c", "This is a string"); ARC.Scripting.VariableManager.SetVariable("$d", 0x055); ```

Single Variables *ARC.Scripting.VariableManager.ClearVariable(string variableName)* Clear the variable and remove it from memory.

ARC.Scripting.VariableManager.ClearVariables() Clear the entire variable memory.

ARC.Scripting.VariableManager.DoesVariableExist(string variableName) Returns a Boolean if the variable has been defined in memory.

ARC.Scripting.VariableManager.DumpVariablesToString() Returns a string with each variable and the corresponding value (one per line). This is similar to the Variable Manager control found in EZ-Builder->Add Control->Scripting->Variable Manager.

ARC.Scripting.VariableManager.GetVariable(string variableName) Get the value stored in the specified variable.

ARC.Scripting.VariableManager.GetVariable(string variableName, int index) Get the value stored in the specified index of the array variable.

ARC.Scripting.VariableManager.SetVariable(string variableName, object value) Set the value into the memory location of the specified variable. If the variable does not exist, this will create the variable and assign the value.

Array Variables *ARC.Scripting.VariableManager.AppendToVariableArray(string variableName, object value)* Append the value to the existing array.

ARC.Scripting.VariableManager.CreateVariableArray(string variableName, byte[] values)
ARC.Scripting.VariableManager.CreateVariableArray(string variableName, string[] values)
ARC.Scripting.VariableManager.CreateVariableArray(string variableName, object defaultValue, int size) Create an array variable and specify the default values. Use AppendToVariableArray() to add more items to this array.

ARC.Scripting.VariableManager.FillVariableArray(string variableName, object defaultValue) If a variable array already exists, this will fill every defined index of the array with the specified value.

ARC.Scripting.VariableManager.GetArraySize(string variableName) Returns an integer that is the index size of the specified array variable.

ARC.Scripting.VariableManager.IsVariableArray(string variableName) Returns a Boolean if the specified variable is an array.

ARC.Scripting.VariableManager.SetVariable(string variableName, object value, int index) Set the value into the index of the specified array variable. The array size must already be defined. If you attempt to set a value to an index outside of the index size, an exception will be thrown.

Monitoring Variable Changes The variable manager has an event which will be raised for any variable changes. If your plugin has a variable that you wish to watch for changes, subscribe to the event. Remember to unsubscribe from the event when your plugin closes, as per the Plugin Compliance section of this tutorial. Below is an example of subscribing, checking for a variable change, and unsubscribing when the plugin closes.

If the variable is an array, the index will be populated with the array index that has changed. If the variable is not an array, the index will equal -1.

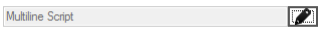
``` private MyForm() {

InitializeComponent();

// Subscribe to variable change event ARC.Scripting.VariableManager.OnVariableChanged += VariableManager_OnVariableChanged; }

private void FormMain_FormClosing(object sender, FormClosingEventArgs e) {

// Unsubscribe to variable change event ARC.Scripting.VariableManager.OnVariableChanged -= VariableManager_OnVariableChanged; }

private void VariableManager_OnVariableChanged(string variableName, object value, int index) {

if (!variableName.Equals("$myVariable", StringComparison.InvariantCultureIgnoreCase)) return;

// Do something because the variable has changed } ```
```

Example: EZ-Script/Blockly Edit Control

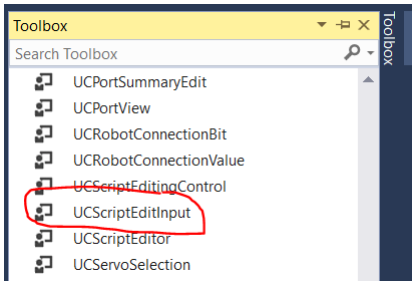
Throughout EZ-Builder controls, you will notice an edit usercontrol which allows single-line, multi-line and blockly editing for EZ-Script. The usercontrol looks like this...

When holding a single line of Script, it looks like this...

Multi-line script is generated by either syntax editor or blockly editor, both of which are accessible by pressing the edit button on the control. When holding multi-lines of script, generated from either Blockly or Syntax editing UI, it looks like this...

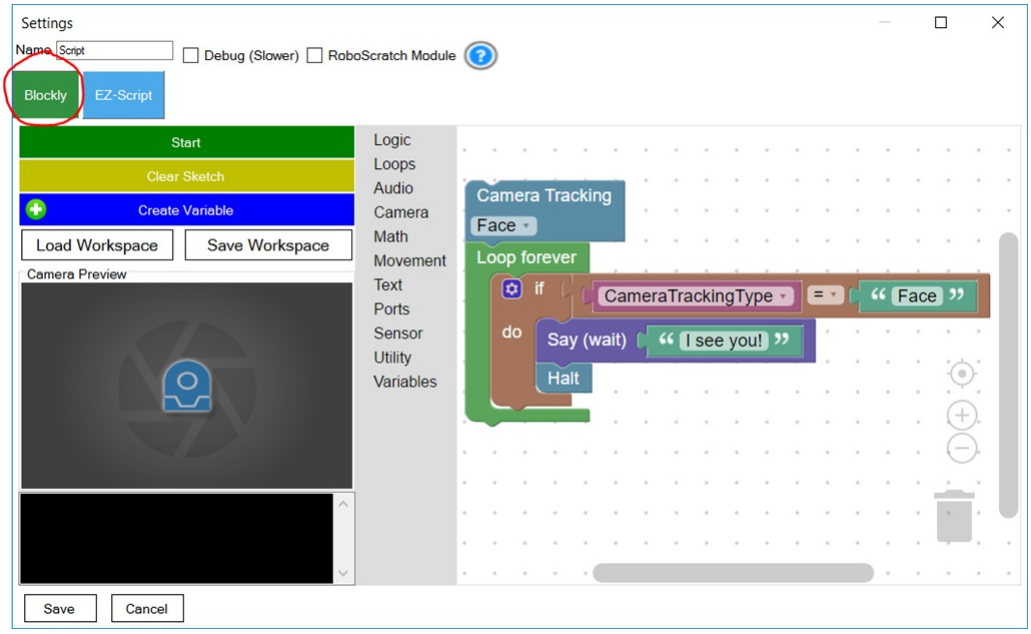


This control can be found as a component within the ARC.exe, and it is called "UCScriptEditInput"

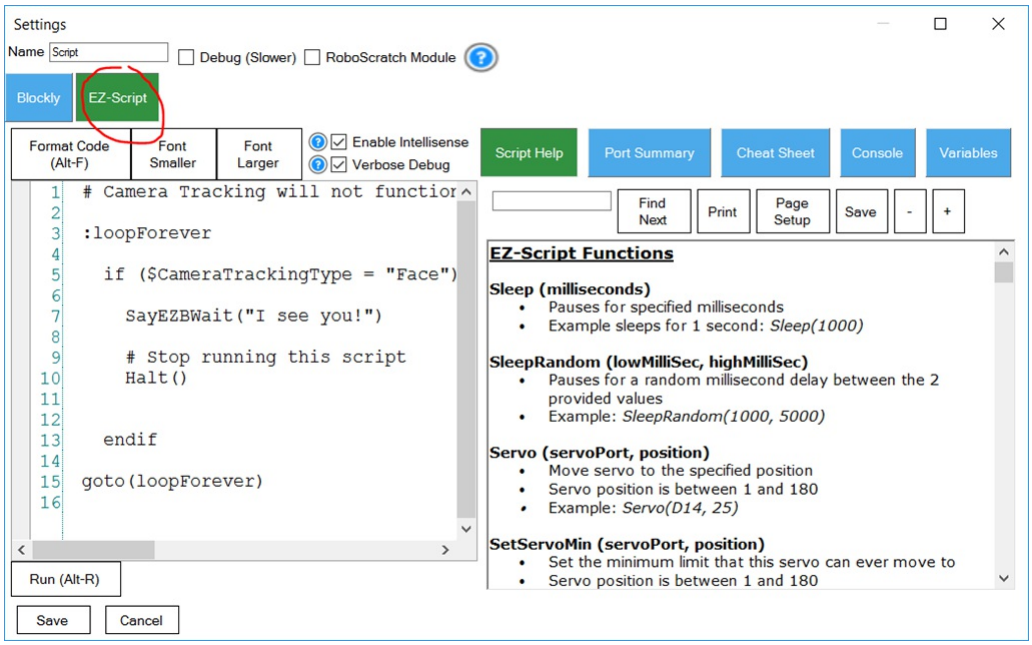


When the edit button of the UCScriptEditInput is pressed, it will display a new window containing both script syntax editor and blockly code creator. The default view (blockly/syntax) is based on 1) the user's preferred editor from preferences menu, 2) the last saved edit mode. ARC will decide which view to display to the user.

Blockly Tab Of Editor



Script Syntax Tab Of Editor



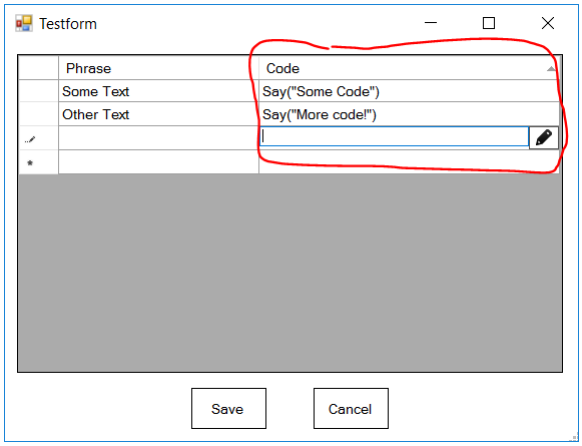
Loading Saved Code Into UCScriptEditInput Your plugin should be saving code that was created by the user to the project file via the cf.STORAGE option demonstrated in an earlier step of this tutorial. The UCScriptEditInputrequires the loaded data in two formats, the VALUE and XML. The VALUE property contains the raw EZ-SCRIPT that was edited using the Syntax Editor. The XML is the Blockly configuration. In the code example below, the UCScriptEditInput is populated by the cf.STORAGE.

```
public void SetConfiguration(PluginV1 cf) {

  ucScriptEdit1.Value = cf.STORAGE["MyCodeValue"].ToString();
  ucScriptEdit1.XML = cf.STORAGE["MyCodeXML"].ToString();
}
```

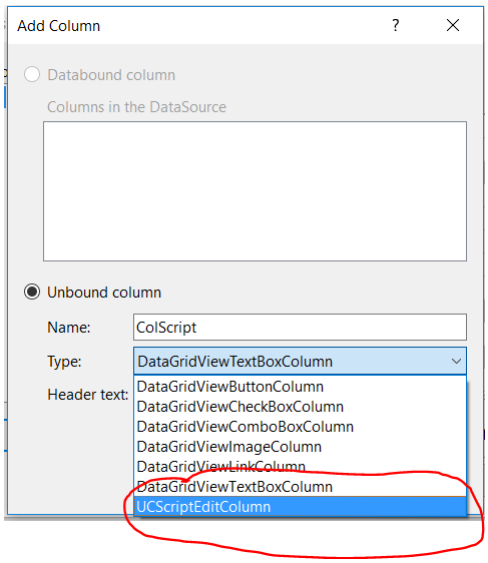
Saving Code from UCScriptEditInput As you saw from the earlier step, the user code is loaded in two parts, the XML and script. Both of those properties contain the data which will be saved to the cf.STORAGE as well.

```
public PluginV1 SaveCode() {

  PluginV1 cf = new PluginV1();

  cf.STORAGE["MyCodeValue"] = ucScriptEdit1.Value;
  cf.STORAGE["MyCodeXML"] = ucScriptEdit1.XML;

  return cf;
}
```

Script Editing in DataGridView Having multiple saved scripts in a DataGridView is possible as well. This allows users to create multiple scripts that could be triggered based on an input, for example. Reference of this behavior is the Speech Recognition or Twitter Recognition controls. They each allow the user to add custom scripts that are triggered on an input (Phrase) value.



When defining an EZ-Script column for the DataGridView, select the UCScriptEditColumn for the type.



Populating Saved Code Loading user saved code from the project configuration file to UCScriptEditColumn is a little different than loading to a single UCScriptEditInput. The difference is that a UCScriptEditColumn accepts the CODE and XML to be provided in a single class (UCForms.UC.FormScriptEdit.ConfigurationCls) and passed to the VALUE property. In this example below, the saved data is stored as a CustomObjectv2 in the project configuration. As you learned in an earlier step of this tutorial, the CustomObjectv2 will save your custom class using reflection into the project file.

The code below will foreach loop through each item of the saved code class, which contains the Phrase, Code and XML of each user added item in the DataGridView. Notice how the Code and XML are populated to the UCScriptEditColumn within the UCForms.UC.FormScriptEdit.ConfigurationCls class.

``` using System.Windows.Forms;

namespace EZ_Builder {

public partial class Testform : Form {

```
public class SavedCodeCls {

  public SavedCodeItemCls [] UserCodes = new SavedCodeItemCls[] { };

  public class SavedCodeItemCls {

    public string Phrase = string.Empty;
    public string Code = string.Empty;
    public string XML = string.Empty;
  }
}

public Testform() {

  InitializeComponent();
}

public void LoadSavedData(Config.Sub.PluginV1 cf) {

  SavedCodeCls savedCode = (SavedCodeCls)cf.GetCustomObjectV2(typeof(SavedCodeCls));

  foreach (var codeItem in savedCode.UserCodes) {

    // Add a new row and get the index of it
    int rowIndex = dataGridView1.Rows.Add();

    // Assign the saved Phrase to column 0
    dataGridView1.Rows[rowIndex].Cells[0].Value = codeItem.Phrase;

    // Assign the saved Code and XML to column 1
    dataGridView1.Rows[rowIndex].Cells[1].Value = new UCForms.UC.FormScriptEdit.ConfigurationCls(codeItem.Code, codeIte
  }
}
```

} } ```

**Saving Code** We will now expand on the above code example to include a function for saving the user defined rows and code from the DataGridView to the project file.

``` using System.Windows.Forms; using System.Collections.Generic;

namespace EZ_Builder {

public partial class Testform : Form {

```
public class SavedCodeCls {

  public SavedCodeItemCls [] UserCodes = new SavedCodeItemCls[] { };

  public class SavedCodeItemCls {
```

```
    public string Phrase = string.Empty;
    public string Code = string.Empty;
    public string XML = string.Empty;
  }
}

public Testform() {

  InitializeComponent();
}

public void LoadSavedData(Config.Sub.PluginV1 cf) {

  SavedCodeCls savedCode = (SavedCodeCls)cf.GetCustomObjectV2(typeof(SavedCodeCls));

  foreach (var codeItem in savedCode.UserCodes) {

    // Add a new row and get the index of it
    int rowIndex = dataGridView1.Rows.Add();

    // Assign the saved Phrase to column 0
    dataGridView1.Rows[rowIndex].Cells[0].Value = codeItem.Phrase;

    // Assign the saved Code and XML to column 1
    dataGridView1.Rows[rowIndex].Cells[1].Value = new UCForms.UC.FormScriptEdit.ConfigurationCls(codeItem.Code, codeIte
  }
}

public Config.Sub.PluginV1 GetSavedData() {

  var cf = new Config.Sub.PluginV1();

  List items = new List();

  foreach (DataGridViewRow dgvr in dataGridView1.Rows) {

    // Check if this row is valid by seeing if any necessary columsn are null
    if (dgvr.Cells[0].Value == null)
      continue;
    else if (dgvr.Cells[1].Value == null)
      continue;

    // Get the values of each column
    var phrase = dgvr.Cells[0].Value.ToString();
    var code = (UCForms.UC.FormScriptEdit.ConfigurationCls)dgvr.Cells[1].Value;

    // Assign the values (code, xml and phrase) to the item
    var codeItem = new SavedCodeCls.SavedCodeItemCls();
    codeItem.Code = code.Code;
    codeItem.XML = code.XML;
    codeItem.Phrase = phrase;

    // Add the item to the list
    items.Add(codeItem);
  }

  // Assign the list of code items to the array within the master clas
  SavedCodeCls savedCode = new SavedCodeCls();
  savedCode.UserCodes = items.ToArray();

  // Add the master config class to the project configuration as a custom object
  cf.SetCustomObjectV2(savedCode);

  return cf;
}
```

}} ```

Example: EZ-Script Executor

EZ-Script is the programming language used in EZ-Builder. The scripting language is similar to Basic and has many robot specific commands. The power of EZ-Script also allows controls to interact with each other using the ControlCommand(), which you may have already read about in this tutorial.

This part of the tutorial will demonstrate how to execute EZ-Script code within your plugin. An example of how this is useful is if your plugin provides configuration for the user to configure custom code for a specific action. If you look at DJ's Tic Tac Toe example, you will notice that the configuration dialog allows the user to create custom EZ-Script for Winning, Losing and Tie.

The namespace for all scripting methods is *EZ_Builder.Scripting*.

Simple Example This code example demonstrates how to run an ez-script piece of code that will speak a phrase out of the speaker when the button is pressed. You will need to add a Button to your form and assign the Click event for it to work.

``` public partial class MainForm : EZ_Builder.UCForms.FormPluginMaster {

```
EZ_Builder.Scripting.Executor _executor;

public MainForm()
  : base() {

  InitializeComponent();

  _executor = new EZ_Builder.Scripting.Executor();
}

private void button1_Click(object sender, EventArgs e) {

  _executor.StartScriptASync("Say(\"Hello, I am ez-script\")");
}
```

} ```

**Example with Events** There are many events that can be associated with the Executor, such as OnDone, OnError, etc.. This allows your program to accommodate behaviors. In this example, the Executor will display a message box when the script has completed executing.

``` public partial class MainForm : EZ_Builder.UCForms.FormPluginMaster {

```
EZ_Builder.Scripting.Executor _executor;

public MainForm()
  : base() {

  InitializeComponent();

  _executor = new EZ_Builder.Scripting.Executor();
  _executor.OnDone += _executor_OnDone;
}

private void button1_Click(object sender, EventArgs e) {

  _executor.StartScriptASync("Say(\"Hello, I am ez-script\")");
}

void _executor_OnDone(string compilerName, TimeSpan timeTook) {

  MessageBox.Show(string.Format("Script has completed and took {0} milliseconds", timeTook.TotalMilliseconds));
}
```

} ```

Starting A New Script When Another Is Running If the executor is currently running an ASync script in the background while you launch another, the first script will be cancelled and the most recent script will begin executing. Each instance of an Executor can run only one script. To run multiple scripts at the same time, use an Executor per script.

Inside the Executor The executor has many methods and events.

EZ_Builder.Scripting.Executor.ExecuteScriptSingleLine(string line); Executes only one line of EZ-Script and blocks until it has completed. This method returns the result of the single line of code. For example, if the code was a function to return the value of a servo (example GetServo(d0)), the value of the servo d0 will be returned.

EZ_Builder.Scripting.Executor.Resume(); There is an EZ-Script command to "Pause" the current running script. If the command is ever executed in the script, this will resume the execution.

EZ_Builder.Scripting.Executor.StartScriptASync(Command[] compiled); This executes the compiled Command Opcode array in the background. If you precompile the script into an array of Command Opcodes, this method can be used. The advantage to pre-compiling the source into Command Opcodes is that the script will execute faster because it will not need to be compiled each time. There is a compiler cache in the Executor, however. This means that if you run the same script twice that is not compiled, the last Opcode cache will be used.

EZ_Builder.Scripting.Executor.StartScriptASync(string script); This method compiles the plain text EZ-Script and executes it in the background. There is a compiler cache in the Executor, however. This means that if you run the same script consecutive times, the last Opcode cache will be used.

EZ_Builder.Scripting.Executor.StartScriptBlocking(Command[] compiled); Executes the compiled Command OpCode array on the current thread.

EZ_Builder.Scripting.Executor.StartScriptBlocking(string script); This method compiles the plain text EZ-Script and executes it on the

current thread. There is a compiler cache in the Executor, however. This means that if you run the same script consecutive times, the last Opcode cache will be used.

EZ_Builder.Scripting.Executor.StopScript(); Stops the current ASync running EZ-Script on this Executor.

Events These events can be assigned to an Executor and will be raised at their appropriate function. The "CompilerName" parameter will include the optional compiler name that can be provided when the Executor class is initiated. In the above examples, the Executor class is created without any parameters. If a parameter was supplied, it would be the name of this compiler. If your program has many Executors that share these events, the CompilerName parameter will come in handy to identify what executor it originated from.

event OnCmdExecHandler(string compilerName, int lineNumber, string execTxt) Event is raised for each line that is executed in the script. This will dramatically slow the execution of the script, but is great for debugging.

event OnDoneHandler(string compilerName, TimeSpan timeTook) Event is raised when the script has completed.

event OnPausedHandler(string compilerName) Event is raised when the EZ-Script calls the Pause method. The Executor.Resume() function is necessary to continue, or StopScript().

event OnResumeHandler(string compilerName) Event is raised when the script is instructed to resume after a Pause.

event OnStartHandler(string compilerName) Event is raised when the script begins to execute.

Example: Custom EZ-Script Function

The EZ-Script engine has two event to add your own custom EZ-Script functions for users to use. The two different events will call before (Override) and after (Additional) the EZ-Script executor. That means you can either override existing commands or add new commands.

Additional Commands Adding a command extends the existing commandset of EZ-Script. The example below demonstrates adding a new command to EZ-Script called "SetColor()". ``` SetColor(20, 100, 20) ```

The *ExpressionEvaluation.FunctionEval.AdditionalFunctionEvent* event is raised. Your plugin may attach to this event and process functions for the script.

Here is an example plugin that creates a user defined function called **SetColor()**: https://synthiam.com/Software/Manual/User-Defined-Function-Example-15864

``` using System; using System.Drawing; using System.Windows.Forms;

namespace User_Defined_Function_Example {

public partial class FormMaster : EZ_Builder.UCForms.FormPluginMaster {

```
public FormMaster() {

  InitializeComponent();
}

private void FormMaster_Load(object sender, EventArgs e) {

  // Intercept all unknown functions called from any ez-script globally.
  // If a function is called that doesn't exist in the ez-script library, this event will execute
  ExpressionEvaluation.FunctionEval.AdditionalFunctionEvent += FunctionEval_AdditionalFunctionEvent;
}

private void FormMaster_FormClosing(object sender, FormClosingEventArgs e) {

  // Disconnect from the function event
  ExpressionEvaluation.FunctionEval.AdditionalFunctionEvent -= FunctionEval_AdditionalFunctionEvent;
}

///
/// This is executed when a function is specified in any ez-scripting that isn't a native function.
/// You can check to see if the function that was called is your function.
/// If it is, do something and return something.
/// If you don't return something, a default value of TRUE is returned.
/// If you throw an exception, the ez-script control will receive the exception and present the error to the user.
///
private void FunctionEval_AdditionalFunctionEvent(object sender, ExpressionEvaluation.AdditionalFunctionEventArgs e) {

  // Check if the function is our function (SetColor)
  if (!e.Name.Equals("setcolor", StringComparison.InvariantCultureIgnoreCase))
    return;

  // Check if the correct number of parameters were passed to this function
  if (e.Parameters.Length != 3)
    throw new Exception("Expects 3 parameters. Usage: SetColor(red [0-255], green [0-255], blue [0-255]). Example: SetC

  // Convert the parameters to datatypes
  byte red = Convert.ToByte(e.Parameters[0]);
  byte green = Convert.ToByte(e.Parameters[1]);
  byte blue = Convert.ToByte(e.Parameters[2]);

  // Do something
  EZ_Builder.Invokers.SetBackColor(label1, Color.FromArgb(red, green, blue));

  // Return something. Good idea to return TRUE if your function isn't meant to return anything
  e.ReturnValue = true;
}
```

} } ```

**Override Commands** Override commands will replace the existing EZ-Script command with your own version. This means you can intercept the command and handle the logic yourself. An example of this functionality is in the Ultrasonic Ping Sensor control, which replaces the GetPing() command with it's own version.

*Note: This event has OverrideFunctionEventArgs() which has a IsHandled(bool) that must be set if you hanlded the return object.

The *ExpressionEvaluation.FunctionEval.OverrideFunctionEvent* event and your logic must exist in there to override the existing command. If you expect a different number of parameters, you can still return and let the main EZ-Script handle it by not setting IsHandled in the OverrideFunctionEventArgs.

``` using ExpressionEvaluation; using EZ_B; using EZ_Builder.Config; using EZ_Builder.Scripting;

namespace EZ_Builder.UCForms {

public partial class FormPing : FormMaster {

```
public FormPing() {

  InitializeComponent();
```

```
  progressBar1.Maximum = EZ_B.HC_SR04.MAX_VALUE;

  // Set the override event
  ExpressionEvaluation.FunctionEval.OverrideFunctionEvent += FunctionEval_OverrideFunctionEvent;
}

private void FormPing_FormClosing(object sender, FormClosingEventArgs e) {

  // detach from the override event
  ExpressionEvaluation.FunctionEval.OverrideFunctionEvent -= FunctionEval_OverrideFunctionEvent;
}

private void FunctionEval_OverrideFunctionEvent(object sender, ExpressionEvaluation.OverrideFunctionEventArgs e) {

  if (e.Name.Equals(OpCodeReadEnum.getping.ToString(), StringComparison.InvariantCultureIgnoreCase)) {

    FunctionEval.CheckParamCount(e.Name, e.Parameters, 2);

    string triggerStr = e.Parameters[0].ToString();
    string echoStr = e.Parameters[1].ToString();
    HelperPortParser cmdTrig = new HelperPortParser(triggerStr);
    HelperPortParser cmdEcho = new HelperPortParser(echoStr);

    if (cmdTrig.DigitalPort != _cf.Ping.PingTriggerPort ||
        cmdTrig.BoardIndex != _cf.Ping.EZBIndex ||
        cmdEcho.DigitalPort != _cf.Ping.PingEchoPort ||
        cmdEcho.BoardIndex != _cf.Ping.EZBIndex)
      return;

    if (!Invokers.GetCheckedValue(cbPause))
      return;

    if (cmdTrig.BoardIndex != cmdEcho.BoardIndex)
      throw new Exception("Trigger and Echo ports must be on the same EZ-B");

    if (!EZBManager.EZBs[cmdTrig.BoardIndex].IsConnected)
      throw new Exception(string.Format("Not connected to EZ_B {0}", cmdTrig.BoardIndex));

    int value = EZBManager.EZBs[cmdTrig.BoardIndex].HC_SR04.GetValue(cmdTrig.DigitalPort, cmdEcho.DigitalPort);

    updateDisplayData(value);

    e.ReturnValue = value;
    e.IsHandled = true;

    return;
  }
}
```

}} ```

Example: Custom Movement Panel

The EZ-Builder software uses controls that register themselves as a movement panel. This allows your plugin to listen to movement requests from other controls. When any control or ez-script calls for a movement direction (i.e. Forward, Left, Stop, etc), your plugin can be responsible for moving the robot. There can only be one movement panel per robot project. This is because there is only one method of locomotion for a robot, which this movement panel would be responsible for.

To understand more about how Movement Panels work in EZ-Builder, read this tutorial.

What's In A Movement Panel? A movement panel will have buttons that lets the user specify directions that the robot should move. Your movement panel will be responsible for the robot moving. This means that anywhere a direction is specified, your control will be responsible for moving the robot. Generally a movement panel has speed controls in the form of trackbars of some sort.

Code Example
Want to make your own movement panel? Here is an example of how to implement code which will respond to movement requests:
```
public FormMain()
    base() {

InitializeComponent();

// Assign this control as a movement panel so the software knows who owns the movements EZ_Builder.FormMain.MovementPanel = this;

// Assign the movement locomotion style for this control. // There are different kind of locomotion for your robot, and this helps other // controls know what to expect when a movement is happening. // Check out the ENUM to see what other locomotion styles there are that // fits your movement panel type. EZBManager.MovementManager.LocomotionStyle = LocomotionStyleEnum.GAIT;

// assign the movement event // this event is raised when another ARC control requests movement
EZBManager.MovementManager.OnMovement2 += Movement_OnMovement2;

// assign the speed change event // this event is raised when another control or user changes the speed
EZBManager.MovementManager.OnSpeedChanged += Movement_OnSpeedChanged; }

private void FormModifiedServoMovementPanel_FormClosing(object sender, FormClosingEventArgs e) {

// Remove this control as a movement panel EZ_Builder.FormMain.MovementPanel = null;

EZBManager.MovementManager.OnSpeedChanged -= Movement_OnSpeedChanged;

EZBManager.MovementManager.OnMovement2 -= Movement_OnMovement2; }

private void Movement_OnSpeedChanged(int speedLeft, int speedRight) {

// do something with the speed change }

private void Movement_OnMovement2(MovementManager.MovementDirectionEnum direction, byte speedLeft, byte speedRight) {

// ** // do something based on the speed
// handle speed change here // **

// Now do something based on the new movement direction

if (direction == MovementManager.MovementDirectionEnum.Forward) {

// handle custom Forward movement

} else if (direction == MovementManager.MovementDirectionEnum.Reverse) {

// handle custom Reverse movement

} else if (direction == MovementManager.MovementDirectionEnum.Right) {

// handle custom Right movement

} else if (direction == MovementManager.MovementDirectionEnum.Left) {

// handle custom Left movement

} else if (direction == MovementManager.MovementDirectionEnum.Stop) {

// handle custom Stop movement

} } ```
```

Example: Output Audio from EZ-B

This is an example with source code about how to play audio out of the EZ-B. The EZ-B will output audio as a stream or byte array.

This example can be run as a plugin by installing it here: https://www.ez-robot.com/EZ-Builder/Plugins/view/202

Source code as a plugin project is here: OutputAudioFromEZ-BSource.zip

*Dependency: Additional to adding EZ-Builder.exe and EZ-B.DLL, this plugin requires NAudio.DLL library to be added as a project reference. Remember to UNSELECT "Copy Files"!

This plugin provides the following examples:

1. Load audio from MP3 or WAV file ``` // MP3 NAudio.Wave.Mp3FileReader mp3 = new NAudio.Wave.Mp3FileReader(openFileDialog1.FileName);

// WAV NAudio.Wave.WaveStream wav = new NAudio.Wave.WaveFileReader(openFileDialog1.FileName); ```

2. Convert audio file to uncompressed PCM data to supported EZ-B sample rate and sample size ``` NAudio.Wave.WaveFormatConversionStream pcm = new NAudio.Wave.WaveFormatConversionStream(new NAudio.Wave.WaveFormat(EZ_B.EZBv4Sound.AUDIO_SAMPLE_BITRATE, 8, 1), mp3); ```

3. Compress PCM data with gzip to be stored in project STORAGE ``` using (MemoryStream ms = new MemoryStream()) {

```
            using (GZipStream gz = new GZipStream(ms, CompressionMode.Compress))
              pcm.CopyTo(gz);

            _cf.STORAGE[ConfigTitles.COMPRESSED_AUDIO_DATA] = ms.ToArray();
          }
```

```

4. Play audio data from compressed project STORAGE ``` using (MemoryStream ms = new MemoryStream(compressedAudioData)) using (GZipStream gz = new GZipStream(ms, CompressionMode.Decompress)) EZBManager.EZBs[0].SoundV4.PlayData(gz); ```

5. Supports ControlCommand() for Play and Stop of audio to be used in external scripts ``` public override object[] GetSupportedControlCommands() {

   List items = new List();

   items.Add(ControlCommands.StartPlayingAudio); items.Add(ControlCommands.StopPlayingAudio);

   return items.ToArray(); }

   public override void SendCommand(string windowCommand, params string[] values) {

   if (windowCommand.Equals(ControlCommands.StartPlayingAudio, StringComparison.InvariantCultureIgnoreCase)) playStoredAudio(); else if (windowCommand.Equals(ControlCommands.StopPlayingAudio, StringComparison.InvariantCultureIgnoreCase)) stopPlaying(); else base.SendCommand(windowCommand, values); } ```

6. Changes the status of the button when audio is playing globally from anywhere in EZ-Builder on EZ-B #0 ``` public FormMain() {

   InitializeComponent();

   EZBManager.EZBs[0].SoundV4.OnStartPlaying += SoundV4_OnStartPlaying; EZBManager.EZBs[0].SoundV4.OnStopPlaying += SoundV4_OnStopPlaying; }

   private void FormMain_FormClosing(object sender, FormClosingEventArgs e) {

   EZBManager.EZBs[0].SoundV4.OnStartPlaying -= SoundV4_OnStartPlaying; EZBManager.EZBs[0].SoundV4.OnStopPlaying -= SoundV4_OnStopPlaying; }

   private void SoundV4_OnStopPlaying() {

   Invokers.SetText(btnPlayAudio, "Play"); }

   private void SoundV4_OnStartPlaying() {

   Invokers.SetText(btnPlayAudio, "Stop"); } ```

**Output Text to Speech** You can output text to speech easily as well, using the following code example... ``` using (MemoryStream s = EZBManager.EZBs[0].SpeechSynth.SayToStream("I am speaking out of the EZ-B)) EZBManager.EZBs[0].SoundV4.PlayData(s); ```
```

Example: Dependencies, Files and Sub Folders

Your plugin may require dependencies, such as images, fonts or other various files.

DLL Files There are two types of DLL files, managed and unmanaged. A managed DLL is the type that can be added as a resource in the .Net project. The unmanaged DLL is a C or C++ library which is used with Interop calls. If you use either of these, you will be aware of which one is being used.

Any managed DLL file that was added as a resource to your project, should be included to be copied as a resource during compile time. Using managed DLL files does not require any path specifications in your code, as the assembly will be loaded when needed. Visual Studio will take care of copying the file, long as it is marked to be copied, during compile, into the root plugin folder. You should not need to worry about managed DLL location, as visual studio will take care of it, but it must be in the same folder as your Plugin DLL.

The unmanaged DLL usage is a different story. When your plugin references an unmanaged DLL, it also specifies the path. If the path is not specified in the interop decleration, the default directory of your plugin is assumed. The unmanaged DLL must be added to your project as a content type, and included to be copied when compiled. This is because unmanaged DLL files are not added as resources in .Net Visual Studio projects, and therefore you must take care of specifying the file to be copied yourself, during compile time. This is easy by simply adding the file to the project as content, and selecting Copy Always from it's property settings.

The source code of an example project of using both unmanaged and managed DLL files can be found in the Oculus Rift Camera plugin, here: https://www.ez-robot.com/EZ-Builder/Plugins/view/59

Files & Sub Folders If your plugin requires to read files from the plugin folder, here's how you can get the path to the plugin root folder, by combing the guid and windows environment settings for the public path. The location of the plugin folder in the Public Documents is defined by the Windows Operating System. The Public Documents and Plugin path is assembled with the method *EZ_Builder.Constants.PLUGINS_FOLDER*, as seen in code examples below.

This code example combines the user's public ARC plugin folder, that comes from the windows environment settings, the guid and the filename.

``` using System.IO;

string OculusRift_Fx_File = EZ_Builder.Common.CombinePath( EZ_Builder.Constants.PLUGINS_FOLDER, _cf._pluginGUID, "OculusRift.fx");

if (!File.Exists(OculusRift_Fx_File)) {

MessageBox.Show("Unable to find the OculusRift.fx file that should have installed with this plugin. Cancelling");

return;

} ```

So if you have a fonts sub folder in the plugin folder, or some sub in the plugin folder, you can do this...

``` string FontFile = EZ_Builder.Common.CombinePath( EZ_Builder.Constants.PLUGINS_FOLDER, _cf._pluginGUID, "Fonts", "SomeFont.ttf"); ```

Or to read the files from a sub folder of the plug directory...

``` using System.IO;

```
  string fontFolder = EZ_Builder.Common.CombinePath(
    EZ_Builder.Constants.PLUGINS_FOLDER,
    _cf._pluginGUID,
    "Fonts");

  foreach (string fontFile in Directory.GetFiles(fontFolder)) {

        Console.WriteLine("File: {0}", fontFile);
  }
```

```

Example: Finding Other Behavior Controls

Overview Similar to the *Example: Camera Control*Â step of this tutorial, you can search for other controls added to the project workspace. There's a number of *EZBManager.FormMain* methods that make it easy. Additionally, there are a few *EZBManager.FormMain* events that allow your plugin to watch for added and removed controls. We'll discuss and provide examples of accessing other controls in the current project.

EZBManager.FormMain.OnProjectLoadCompletedï»¿() This event is raised when a project has completed loading all controls to the workspace. This is a handy method if you're looking for a control when the project is loaded. This event will ensure all other behavior controls have loaded before raising.Â Event raised after a project has completely loaded all behavior controls. If you're control is looking to bind to another control, find the control in this event. If you attempt to look for a control (i.e. camera) during constructor or SetConfiguration, the other control may not have loaded from the config yet. This event is raised after all of the controls have been loaded to the workspace

EZBManager.FormMain.OnBehaviorControlAdded(object newControl, int page) Event raised when a new control is added to the project workspace. The control and virtual desktop page will be returned.Â Event raised when a control is added to the workspace. This could be during the project load event, or if a user uses the Add Control menu. If you're wanting to keep track of new controls added to the workspace, this is how to do it. However, if you're expecting a control to exist when a project is loaded, look into OnProjectLoadCompleted event.

EZBManager.FormMain.OnBehaviorControlRemovedHandler(object removedControl) Event raised when an existing control is being removed from the project workspace. This is not raised when the application is closing or the project is closing.Â Contrary to OnBehaviorControlAdded, this event is raised when a control is removed by a user from the current workspace. The control is passed as a parameter, but it won't be actually closed until you release it from the event completing. This means don't expect the control to exist once your method attached to this event has completed.

Plugin Compliance

Overview EZ-Robot is committed to offer users a secure and dependable robot development environment. Due to the high expectation of efficient, dependable and secure from users, there are a few dependencies and restrictions which we enforce in your plugins during review.

Avoid Length GUI Thread Processing It is very convenient to throw a bunch of code into an event raised by a GUI widget, such as a button or checkbox. When code is executed in an event raised by the user interface, it runs in the thread of that object. This means lengthy code will delay/pause the user interface experience until the code has completed and processing is returned to the GUI thread. This behavior must be reduced at all cost by using events, threading or background workers.

Events .Net framework provides a number of events for controls, forms and such. Additionally, the EZ-Builder framework provides events for various activities. It is highly preferred to use Events rather than Timers. This includes servo movements, new camera frames, adding/removing controls to workspace, and more. It's goodÂ etiquette to unsubscribe from events when your plugin is being closed in the OnClosing() event. If you do not unsubscribe from events, the .Net framework will not know your plugin and respective controls have been disposed, and therefore it may still attempt to execute the method. Notice that an unsubscribe from event is a -= and a subscribe to event is += ``` public MyPluginForm() {

// Subscribe to events to monitor control activity on the workspace OnBehaviorControlAdded += FormMain_OnBehaviorControlAdded; OnBehaviorControlRemoved += FormMain_OnBehaviorControlRemoved; }

private void FormMain_OnBehaviorControlAdded(object newControl, int page) {

MessageBox(string.Format("{0} has been added to the workspace #{1}", newControl.Text, page)); }

private void FormMain_OnBehaviorControlRemoved(object removedControl) {

MessageBox(string.Format("{0} has been removed from the workspace", removedControl.Text); }

private void MyPluginForm_FormClosing(object sender, FormClosingEventArgs e) {

// Unsubscribe from events that I subscribed to while my plugin is going away OnBehaviorControlAdded -= FormMain_OnBehaviorControlAdded; OnBehaviorControlRemoved -= FormMain_OnBehaviorControlRemoved; } ```

Timers One of the most common timers that is used from convenience is *System.Windows.Forms.Timer*, which is heavily frowned upon and will always have your plugin revoked from public access. An alternative and accepted timer for your background worker is *System.Timers.Timer*.

The difference between these two timers is trivial programatically, but vast in their operational behavior. The System.Windows.Forms.Timer will raise the elapsed event in the user interface thread, while the System.Timers.Timer will raise the event in a background thread.

As a new programmer, you may be familiar with the behavior of System.Windows.Forms.Timer not raising the elapsed event until the previous event has completed. With System.Timers.Timer, if your last elapsed event has not completed, a new one will still be raised. Avoid using a Lock() statement for this behavior, and instead use a boolean variable shown in this example...

``` System.Timers.Timer _timer; bool _isRunning = false; // variable to ensure timer runs once at a time

```
public FormMaster() {

  InitializeComponent();

  _timer = new System.Timers.Timer();
  _timer.Elapsed += _timer_Elapsed;
  _timer.Interval = 100;
  _timer.Start();
}

void _timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e) {

     // Check if another copy of the time event is running, if so get out
  if (_isRunning)
    return;

  _isRunning = true;

  try {

    // Do some work
  } catch (Exception ex) {

    // Uh oh!
  } finally {

    _isRunning = false;
  }
}
```

```

Cross-Thread Invoking While new programmers may feel comfort placing code in events owned by GUI widgets in the user interface thread, there is a different experience when working in a background thread. A background thread will not be able to modify parameters of a GUI object that exists on a different thread. This is called a Cross-Threading Exception. EZ-Builder has a helper class to make life easy for you, which is *EZ_Builder.Invokers*. You will need to use Invokers when updating UI components from System.Timers.Timer or most events that aren't triggered by UI. The Invokers class will check if an invoke is required.

``` void _timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e) {

```
  if (_isRunning)
```
```

```
    return;

  _isRunning = true;

  try {

    EZ_Builder.Invokers.SetText(textbox1, "Here is a number: {0}", 5);

    EZ_Builder.Invokers.SetChecked(checkbox1, false);

    EZ_Builder.Invokers.SetBackColor(button1, System.Drawing.Color.Red);
  } catch (Exception ex) {

    // Uh oh!
  } finally {

    _isRunning = false;
  }
}
```

User Configuration Settings Never under any circumstances save user configuration settings in a separate project file. Always save user configuration settings in the EZ-Builder Project File using the tutorial step for Saving/Loading Configuration. Plugins that save configuration data locally to the drive will be revoked from public status. This is to ensure a seamless user experience within the EZ-Builder environment. If you have questions about saving custom user data, please inquire on the community forum for assistance.

Exception Handling Always wrap code in Try Catch to avoid unhandled exceptions, which will exit EZ-Builder. Your plugin will execute under the EZ-Builder master assembly. If your plugin throws an error that is not handled within an exception, the EZ-Builder instance may close.

```
try {

    // do some work

  } catch (Exception ex) {

    EZ_Builder.EZBManager.Log("Error in control '{0}'. Message: {1}", this.Text, ex.Message);
  }
```