= Argon =

== Introduction ==

Argon allows game designers to configure Unity components and Unity-specific options from within Blender.

Argon has two parts:

- **a Blender addon:** which exports FBX files marked up with custom properties
- **a Unity plugin:** which imports those files and applies their properties as Unity components.

The import script also sets up connections between components as needed.

== Installation ==

- Install the package using the Unity Package Manager
- Open Blender and open a Preferences view
- Go to Add-ons
- Click the ‘Install…’ button
- Find and install Argon_Plugin.zip under Resources/Blender
- The Argon plugin activation box should come into view (possibly after several seconds)
- Click the check box to activate it
- You should see a new tab on the right side of your 3D views in Blender that is titled ‘Argon’ (Press ‘n’ if you don’t see any right side tabs)

== Setup the Default Player ==

Argon provides a default player prefab that demonstrates how to use Argon's runtime components. Find the default player prefab in argon > DefaultPlayer. To use the prefab:

- Drag it into your scene
- Go to the Tools menu > Argon > Demo Scene > Restore Demo Scene Input Manager

== Adding Properties to Objects ==

- In Blender, select an object or objects.

* Use the ‘Add key to selected’ drop down menu to add a property.

- With the property selected, configure it as needed.

= Kinds of Properties =

Some properties match Unity components exactly.

For example, a rigidbody property applies a Rigidbody component in Unity.

For each object with this kind of property, the import script will apply the equivalent Unity component.

Other properties, like ‘tag’, ‘static flags’, and ‘layer’, apply configurations on the imported GameObject.

E.g. adding a tag property will apply that tag on the imported GameObject.

Still other properties, e.g. ‘no renderer’, ‘replace with prefab’, and ‘destroy’, act as import directives.

For example, adding a ‘no renderer’ property will cause the import script to remove the GameObject’s renderer component.

Properties like ‘Interaction Handler’, ‘Playable Scalar Adapter’ and ‘Spawner’ apply one or more Argon specific components.

= Editing Properties: =

- Select the object whose property you’d like to configure.
- Find the list containing that object’s properties, below the ‘Add key to selected’ drop down.
- Select the property.
- Edit the property in the configuration panel.

= Exporting to Unity: =

- Click the Export button in the Argon tab and export an FBX to your Unity project
- In Unity, right-click the file and choose 'Reimport'
- Drag the FBX file into a scene in Unity to verify that the resulting asset has all of the components and modifications that were specified in the Blender file

= Properties: =

== Audio Enable ==

Add an AudioEnable Component.

== Box Collider ==

Add a Box Collider. The dimensions of the box will fit the object.

== Cam Lock Session Enable ==

Add a Cam Lock Session Enable component.

Adding this component to a camera object in Blender is recommended; the import script will add a Camera component if it doesn’t find one attached.

Adding an Re2PickSession on the same object is also recommended: CamLockSessionEnable needs to be next to an IPlayerInteractionTask–and the only implementer of IPlayerInteractionTask is Re2PickSession.

== Component By Name ==

Adds the specified component.

== Destroy ==

Destroy this game object and optionally its children during import.

== Disable Component ==

Disable the specified component during import.

== Enable Receiver ==

Adds an Enable Receiver component.

== Force Pcw ==

Force the importer to generate PlayableClipWrappers for any animation clip whose name contains this component’s name. This is useful in cases where you’re targeting PlayableClipWrappers in your own scripts.

== Interaction Handler ==

Adds either a '''TriggerInteractionHandler''' or a '''ClickInteractionHandler''' to the imported GameObject.

Both trigger and click handlers invoke one or more '''commands''' (see also [[#Commands|Commands]]).

TriggerInteractionHandlers invoke commands on trigger enter and/or exit.

ClickInteractionHandler invoke commands when they receive a click.

You can create commands by clicking the 'New command' button (located in each row of the handler's commands list).

You can also create commands using the Commands panel.

Choose trigger or click with the Type drop down.

=== Trigger Handler Options ===

- Choose to respond to TriggerEnter, TriggerExit or both.
- Enter Signal Value: specify the signal value to send to commands on enter
- Exit Signal Value: specify the signal value to send to commands on exit

=== Click Handler Options ===

- Is Click Hold: If true, respond to mouse down and mouse up. Otherwise, respond to mouse down only.
- Discrete Clicks Also: If '''Is Click Hold''' is true, defines how regular instantaneous clicks should be handled. If false, ignore instantaneous clicks; require at least a brief mouse hold interval.
  If true, treat clicks like mouse hold events but add an artificial hold time between mouse down and mouse up; i.e. when there is a click, invoke commands with a mouse down signal; then wait for ‘Discrete Click Hold Time’ seconds; then invoke commands with a mouse up signal.
- Discrete Click Hold Time: The artificial mouse hold time in the case where a Click-Hold handler receives a click.
- Mouse Down Signal: specify the signal value to send to commands on mouse down
- Mouse Up Signal: specify the signal value to send to commands on mouse up

=== Shared Options (Trigger and Click handlers) ===

- Sleep Also: when this handler goes to sleep, should its attached Collider also be disabled.
- Initial Sleep State: should this handler go to sleep on Start.

== Interaction Highlighter ==

Adds an Interaction Highlighter component.

Mode: Choose the kind of highlighter: '''Highlight Material''', '''Click Beacon''', or '''Invisible'''

=== Highlight Material Options: ===

- Highlight Material: Specify the name of the material to swap to.

=== Click Beacon Options: ===

- Click Beacon Prefab: Specify the name of the prefab to use as a click beacon. There is no need to include the file extension. The prefab must have a component attached to its root object that implements '''IBeaconDevice'''.
- Beacon Placement Option: Specify how to center the beacon: object center or bounds center.
- Beacon Nudge: Defines how far to offset the beacon from its center. In Unity space.
- Rotate Ninety: If true, turn the beacon ninety degrees.
- Downtime Seconds: How long, in seconds, to disable the highlighter following an interaction
- On Sleep Action: When the highlighter Sleeps, should it turn off visually or do nothing.
- Is Invisible to Proximity: Highlighters are automatically found and toggled visible/active/off by Argon's proximity detection system–but not if this is set to true. If true, the detection system will ignore this highlighter. Use when you want to interact with the highlighters in some other way: for example, highlighters that should only activate during a cam lock session.

== Layer ==

Specify a layer to assign to the game object.

== Layer Cam Lock Pickable ==

Assign the game object to the layer “CamLockPickable”. Adding this property is equivalent to adding a Layer property and assigning “CamLockPickable” in the layer field.

== Mesh Collider ==

Apply a mesh collider to this game object

== No Renderer ==

Remove this game object's Renderer component.

This is useful for objects that you would like to see in Blender but not in your scene in Unity; for example, parent objects or collider geometry.

== Object Enable ==

Add an object enable component.

== Off Mesh Link ==

Add an OffMeshLink

== Particle System ==

Add the specified particle system prefab to the imported scene and attach a ParticleSystemEnable component.

- Toggle Game Object: if true, the game object will be set active/inactive when the particle system starts/stops. Don’t set otherwise.

== Playable Scalar Adapter ==

Add a PlayableScalarAdapter component to the game object.

PlayableScalarAdapters set the playback position of a '''playable'''–an animated game object with an associated PlayableClipWrapper–based on an input signal.

In most cases, the signal will come from a Signal Command that targets the object (see [[#Commands|Commands]]).

- Target: Defines the object to play back. This object should have an animation and that animation should be defined within a Blender action (in most cases).
- Is Clip Name Specified Manually: If true, the import script will look for an animation clip whose name matches ‘ClipName’.
  If false, the import script will look for an animation clip whose name includes matches ‘TargetName|ActionName’.
- Clip Name: If Is Clip Name Specified Manually is true. Defines the name of an animation clip to use when constructing the playable during import.
- Action: If Is Clip Name Specified Manually is false. Defines the action whose name should be included when searching for the animation clip to use when constructing this playable during import.

== Re2 Pick Session ==

Add an Re2PickSession component.

Re2PickSessions can handle interaction during a CamLockSession.

Re2PickSession implements IPlayerInteractionTask.

CamLockSessions needs to be adjacent to an IPlayerInteractionTask component.

If you add CamLockSessionEnable to a camera, always add Re2PickSession to the same camera, (unless you’re planning to add a custom component that implements IPlayerInteractionTask; which you could do on an instance of the model in the scene).

- Should Call Click Handlers: If true, the pick session will try to find a '''ClickInteractionHandler''' on the hovered-over object when there is a mouse click. Adding click interaction handlers to objects that the camera can see and setting '''Is Invisible to Proximity''' to '''true''' is the recommended way to implement interaction during a cam lock sessions.
- Try Inventory: If true, the session will try to add any clicked-on objects to the inventory.

== Replace With Prefab ==

Replace this object with the specified prefab. No need to include the “.prefab” extension.

== Rigidbody ==

Add a rigidbody component.

== Screen Overlay Enable ==

Add a Screen Overaly Enable component TODO: decide. not sure this should exists. Isn’t it obsoleted by the command version?

== Slider Collider ==

Add a Slider Collider component.

A Slider Collider sends a signal to its target when it receives an OnTriggerStay method call.

The value of the signal is based on the other collider’s position within the attached collider.

The position is calculated in the Slider Collider’s local space.

The Slider Collider must be attached to a game object with a BoxCollider and that BoxCollider should probably be a trigger collider (otherwise, how will anything enter inside of it).

- Target Type: Defines the type of target to use. (Currently, Animation is the only supported type.)
- Target: Specify the object whose animation should be targeted by the slider collider
- Action: Specify the action on the target object that should be targeted by the slider collider
- Unity Axis: Defines the axis against which to measure the slider value; in Unity space; in the collider’s local space.
- Invert: If true, the resulting signal will be set to one minus itself before it is sent to its target.

Use Slider Colliders to implement objects that animate according to (for example) the player’s movement. See the sliding doors in the demo scene. Use Layers and the Layer Collision Matrix in your project settings, if you need to prevent projectiles from interacting with SliderColliders.

== Spawner ==

Adds a Spawner component.

- Prefab name: specify the prefab to spawn.
- Spawn Mode: defines the spawn mode.
- Spawn Interval: defines the interval in seconds between spawns.

== Static Flags ==

Set the static flags on this object during import.

== Swap Material Enable ==

Add a SwapMaterialEnable component Material Name: the material to swap to.

== Tag ==

Set the tag of this object during import. The tag will be created if it doesn’t already exist.

== Text Mesh ==

Add a Text Mesh or a Text Mesh Pro component

== Visual Effect ==

Add a Visual Effect component. TODO: decide if this and particle system are actually helpful. Should we actually call them ParticleSystemEnable and VisualEffectEnable? Since that would be more accurate?

= Enableable Properties =

Several properties add enableable components to their attached game objects.

Enableable properties include: object enable, component enable, audio enable, particle system enable, screen overlay enable.

These components receive signals (from Send Signal commands) and interpret those signals by turning something on or off.

For example, '''audio enable''' turns an audio clip on or off; '''component enable''' enables or disables a component.

Enableable properties share configuration options under the heading ‘Enable Filters’.

The options are:

- Self Toggle: Toggle on or off with each incoming signal. Ignore the value of the incoming signal.
- Clamp01: Clamp the value of the incoming signal between zero and one.
- Invert: If true, set the value of the incoming signal to one minus the signal before comparing with the threshold value
- Threshold: Defines the value to compare against when evaluating the signal: enable if threshold < signal

= Commands =

Commands define what happens in response to click and trigger events.

To create a new command:

click the ‘+’ button in the Commands panel or click ‘Create command’ in the configuration panel for an Interaction Handler.

Click ‘-’ in the Commands panel to delete a command.

== Animation ==

Play an animation clip on a game object.

- Targets: Defines the list of objects to animate
- Action Name: Defines the name of the action whose animation should be targeted. The import script will search for animation clips whose name matches: “TargetName|ActionName” for each target when building this command.
- Audio Clip Name: name of an audio clip to play during animation. The clip must exist in your project. No need to include the file extension.
- Behaviour: Defines where the animation starts from and how it plays back ** Restart Forwards: always start from the beginning and play forwards ** Toggle and Restart: start from the end and play backwards if the animation played forwards last time. Otherwise start from the beginning and play forwards. ** Flip Directions: start from the current playback position but toggle the playback direction.
- Audio Always Forwards: Always play the audio forwards even when the animation is playing backwards.
- Allows Interrupts: If false, ignore invocations that are received during playback. Don’t ignore otherwise.

== Camera Shake ==

Shake the camera.

- Duration: defines the length of time in seconds to shake the camera
- Displacement Distance: the max distance in Unity meters to move the camera off center

== Command Group ==

A command that invokes a group of other commands.

- Sequential: If true, invoke commands one at a time. If false, invoke all the commands at once

== Display Headline ==

Overlay text on the screen.

- Text: Defines the headline’s text
- Display Time Seconds: Defines the duration of the headline

== Event Only ==

This command broadcasts an event when it is invoked and that is all.

The name of the event is OnCommandEvent.

== Looping Animation ==

Play a looping animation that targets a game object. See the Animation command for documentation of these properties.

== Message Bus ==

- Send a message on the message bus. Messages consist of a string ‘Type’ and a list of target objects. ** '''Type''': Defines a string Type for the message. ** '''Targets''': Defines a list of targets.

== Screen Overlay ==

- Show the specified UI element. ** '''Overlay Name''': Defines the name of the overlay. This must match the name of a ui element in your scene’s UI Document. ** '''Has Duration''': If true, display the overlay for a certain time interval and then hide it again. If false, turn the overlay on indefinitely (or off indefinitely if the signal is less than threshold)” ** '''Duration Seconds''': The length of the interval in seconds when the overlay will show.

== Send Destroy Signal ==

- Destroy the specified targets. Target components that implement ICustomDestroyMessageReceiver will get callbacks before GameObject.Destroy is called on their objects. ** '''Targets''': the targets to destroy

== Send Signal ==

- Send a signal to the specified targets. Use this command to turn enableables on or off. You can also send signals over time to playable scalar adapters as an alternate method for animating objects. ** '''Targets''': A list of target objects. Each target should be able to receive a signal; should have at least one component that implements ISignalHandler. Any object with an enableable–object enable, component enable, audio enable, particle system enable–can receive signals. Sending a signal to an enableable is the recommended way to turn enableables on and off. ** '''Include Children''': If true, the import script will search children of the target object (along with the target object itself) when looking for ISignalHandler components. If false, search the target object only. ** '''Over Time''': If true, send several signals over a given time interval. *** '''Over Time''' Function: Defines the function of time to use when calculating the value to send. **** Start Value End Value: Send the low value at the beginning of the interval and the high value at the end of the interval. **** Saw Tooth: Send values according to a saw tooth function: f(time) = (time % Period) / Period * (HighValue - LowValue) + LowValue. **** Linear: Send value according to a linear function: f(time) = time * (HighValue - LowValue) / Period + LowValue. *** '''Run Indefinitely''' : If true, the signal will broadcast for an indefinite length of time. If false, the signal will stop after the specified interval *** '''Pick-up from Last State''': If true, signal commands will pick up where they left off during the last broadcast. In other words, the first signal sent will equal the last signal sent during the previous invocation of the command. *** '''Duration Seconds''': Defines the length of time in seconds during which the signal will broadcast. *** '''Low Value''': defines the low signal value. *** '''High Value''': defines the high signal value. *** '''Period Seconds''': Defines the period in seconds. For perdiodic functions; e.g. SawTooth. *** '''broadcastIntervalSeconds''': Defines the tick resolution; the amount of time to wait between broadcasts. In seconds. *** '''Clamp Function Output''': If true, clamp the function between low value and high value. *** '''Outro''': Specify the type of outro to use. Outros are a final broadcast that is sent by the command after the main broadcast finishes. **** '''None''': No outro. **** '''Constant''': Send a final broadcast that lerps over a brief interval from the last value sent to Outro Destination Value. The idea is to use this to force the target to return to a certain state once the user is done interacting with it. For example, holding down a button with mouse down; when the mouse is released, the button’s animation should return to the first frame (where it is un-pressed)–so the broadcast should outro to zero. **** '''Threshold Condition''': Send a final broadcast that lerps to either Over

Threshold Destination or Under Threshold Destination depending on whether the last value sent was above or below threshold. For example, this could be used to implement a treasure chest that opens over time when you hold the mouse down; if you don't hold down long enough the chest closes again; if you do, it stays open. *** '''Outro Threshold''': defines the threshold for a Threshold Condition outro *** '''Outro Destination Value''': defines the outro destination value for a Constant type outro. *** '''Over Threshold Destination''': defines the outro destination for when the threshold is met in a Threshold Condition outro. *** '''Under Threshold Destrination''': defines the outro destination for when the threshold is not met in a Threshold Condition outro. *** '''Outro Speed Multiplier''': a constant the speeds up or slows down the outro relative to the speed of the main broadcast.

== Send Sleep/Wake-up Signal ==

Send a sleep or wake-up signal to any components attached to the target(s) that can sleep and wake up. (Targets that can sleep or wake up implement the `ISleep` interface.)

These Include Trigger Interaction Handler, Click Interaction Handler, Interaction Highlighter and any Enableable component: object enable, component enable, particle system enable, etc.. A component that's asleep is unresponsive. Enableables that are asleep aren't necessarily off also. For example, a particle system enable's particle system may still be running while its asleep; its being asleep just means that it won't respond to incoming on/off signals.

== Wait Seconds ==

Wait for some seconds. Use this command to delay the invocation of other commands. You can insert Wait Seconds commands into sequential command groups or link to other commands in the Play After field of a Wait Seconds command.

== Common Command properties: ==

- Signal Filter: defines how to modify the incoming signal: ** Don't Filter: do nothing to the incoming signal. ** Constant Value: replace the incoming value with the value this constant value ** One Minus Signal: set the signal to one minus the signal.
- Play After: Specify a command to play after this one finishes.
- Delay in Seconds: Defines the time in seconds to delay before playing the next command.
- Play After Defer to Latest: If true, the second command will only be invoked at the end of the last delay, in the case where multiple invocations of the first command create overlapping delay intervals. If false, the second command will always fire after delay, ignoring subsequent invocations of the first command.

= Export =

Use this panel to export your scene as an FBX with Argon custom properties. Export your file using this panel only. Exporting an FBX using File > Export > FBX is not supported–many features of Argon will still work but not all. Correct rotation: If true, use EdyJ's export script to fix rotations before exporting. This sometimes fixes rotation issues in the file. Use on a case-by-case basis.

= Material Map =

Add entries to the Material Map panel to conveniently map Blender materials to materials in your Unity project. For each entry choose one of your Blender materials and enter the name of a material in your Unity Project (no need to include ‘.mat’). Use this as a more flexible alternative to the Unity FBX importer’s ‘On Demand Remap’.