The Adventure Toolkit
Reference Manual

Version 1.5 – Sept 2017

Contents

Introduction

Congratulations on purchasing the Adventure Toolkit for Unity! The tools available to you now will provide access to the most commonly needed game functions and dynamics for your games. With them, anyone can build a fully functional game prototype with little or no custom programming required.

For artists and designers, this allows the game to come together in a way that best shows off your work, without having to hire a programmer, or spend time building your own codebase. For programmers, this toolkit can easily serve as an accompaniment to your work; either as an extension of your code or as a safe and reliable way to allow non-programmers on your team to contribute game functionality and dynamics that work alongside your code without interference.

These tools are designed to be flexible, fast and easy-to-use. This reference is here to help you see the tools in detail; their description, their purpose and their properties. You will want to continue to refer to this manual as you become accustomed to the toolkit, and how to implement the variety of dynamics they offer for your game development.

To get started right away, follow the installation instructions found in the ReadMe file. Then, open Unity to confirm the availability of the Adventure Toolkit, under the Component menu. At that point you're ready to start making tools. To make a tool, simply create a new game object and add one of the tool components to it. From there, you can configure the tool to your liking in the Inspector panel.

Tools work as long as they are active and the tool components are enabled. You'll know the tool object is active by the non-grey appearance of the object in the Hierarchy panel. You'll know the tool component is enabled by the check mark found in the top left corner of the Inspector panel, when the tool is selected. This simple method of tools working when active and enabled means that they can be toggled active, or 'triggered', to function at meaningful times. This method follows standard procedure for stock tools available in Unity, such as the Activate Trigger tool, and will work seamlessly alongside them. *Note: Tools that are children of disabled objects are automatically disabled. In cases such as this, make use of enabling disabled components to trigger the tools, in order to keep parent objects active.*

Multiple tools, each handling different tasks, can refer to each other and trigger each other, to form more and more complex constructs called trigger systems. These trigger systems can be rather simple, to control something like a gate opening after a player walks into a collision object, or very complex, such as one to control a full cinematic sequence with effects, dialog and character animations. Because the tools are modular, they work together seamlessly, and because they are atomic, and only handle one specific task or dynamic, when working together in a trigger system they are endlessly flexible.

While the primary focus of this toolkit is to provide the most common dynamics found in popular action adventure games, they can be used in any game or simulation where you want to control various actions, audio, cinematics, events, interactions and scenes.

After you begin experimenting with these tools, you will find them to be helpful, easy-to-use, reliable, flexible, and a joy to work with. So, get started and have fun making games!

How To Use This Reference

This reference contains information on each tool in the Adventure Toolkit. The tools are listed in groups here to match the groups found in Unity. Those tool groups are Action, Audio, Cinematic, Event, Interaction and Scene. For an alphabetical listing of the tools, see the Tool Index at the end of this reference.

The information on each tool includes the tool name, description and purpose as well as details on each editable property you will see in the Inspector panel. The tool description will detail the functionality of the tool, while tool purpose will give a general usage description, often with example case uses.

Property details include the name of the property, the type of data the property represents, and a formal description of the property. Below is a table of some common data types used in this reference, although there are many others, including *Material*, *Animation Curve*, *Audio Clip*, references to other tool components, etc.

Data type	Full name	Example of data type
bool	Boolean	Only True or False
int	Integer	3, 472, -10, etc.
float	Floating-point number	0.0, 12.4, -5.326, etc.
string	Alphanumeric character string	‘cat’, ‘dog123’, ‘bark, bark.’ (without quotes)
GO	Game Object	- A game object reference in the scene or project -
Color	Color (Red, Green, Blue, Alpha)	- Any RGBA color value -
Vector3	Three-dimensional vector (X,Y,Z)	X:0.3,Y:-4.1,Z:2.5 (note that each value is a *float*)

Some tools use an array (or series) of properties grouped into classes. Class properties will be listed with indentations to easily identify them from the other properties. After the class is defined, it will be listed as the data type for another property in that tool.

As an example, the tool Input Relay uses an array of data called ‘inputs’, in which each element of the array is a group of properties, a class, called InputPair. When the data type is listed for the property ‘inputs’, it includes square brackets to indicate this is an array of elements.

InputPair	*class*
someProperty	*dataType*
anotherProperty	*dataType*
…	
inputs	*InputPair[]*

Some tools use properties with a set number of options called an enumerator, or enum, which will appear as a pull-down menu in the Inspector. Like classes, enums will also be used as the data type for some property of that tool.

Aim Lock

Tool: at_AimLock

Description:

Keeps the game object oriented towards a given target. Option to specify the ‘up’ vector.

Purpose:

To aim an object such as a camera at a particular target object.

Properties:

lookAtTarget *GO*

The target object whose position will be the aim of this object at runtime.

upVector *Vector3*

The direction expressing upwards with respect to freedom of rotational movement on this aim lock. (Default: Vector3.up)

useTurnSpeed *bool*

If true, this object will rotate towards the Look At Target based on the Turn Speed. If false, the rotations will happen instantly.

turnSpeed *float*

The rate at which this object will rotate towards the Look At Target, expressed as radians per second. This value is ignored if Use Turn Speed is false.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

Animation Series

Tool: at_AnimationSeries

Description:

Provides an animation schedule for the given game object. Can specify animations, timing, loops, blending and optional game object to activate upon animation completion or series completion.

Purpose:

To make a character or other animation-enabled object play animations in sequence at runtime.

Properties:

AnimStep *class*

stepLabel *string*

Optional name of this animation step. This will overwrite the generic 'Element' default label.

animClip *AnimationClip*

The animation clip to use during this animation step.

noBlend *bool*

If true, will switch to the next animation step abruptly. If false, will cross fade in transition to the next animation step.

loops *int*

The number of times to repeat the animation clip.

activateOnStep *GO*

Optional object to activate upon start of this animation step.

targetGO *GO*

The object whose Animation component will drive this animation sequence.

animSeries *AnimStep[]*

The list of steps in this animation sequence.

defaultIdle *AnimationClip*

Optional looping animation clip to blend to and from during delays and after the sequence is complete.

startDelay *float*

Time in seconds from activation of this tool until the sequence starts.

maxStartDelay *float*

An optional Start Delay defining an upper limit, a range to choose from randomly.

continuousLoop *bool*

If true, will repeat the animation sequence once it has completed.

activateOnComplete *GO*

Optional object to activate upon completion of the animation sequence.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the series. If false, this component will be disabled instead. However, if Continuous Loop is true, this control will be ignored.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

Light Fader

Tool: at_LightFader

Description:

Fades light intensity values on given light game objects either up or down. Options include control over interpolation via animation curve.

Purpose:

To fade lights up or down, or to cross-fade lighting.

Properties:

FadeGroup *class*

eventLabel *string*

Optional name of this fade event. This will overwrite the generic 'Element' default label.

lights *GO[]*

A list of lights to fade as a single group.

targetIntensity *float*

The Light Intensity this group will fade to. Each Light Object in this group will fade from the same intensity value, that of the first Light Object. Typically, lights should therefore be grouped if they are intended to share the same intensity value when this Fade Event begins.

delay *float*

Time in seconds from beginning of this Fade Event until the fade starts.

duration *float*

The duration of the fade in seconds.

fadeInterpolation *AnimationCurve*

Optional rate of fade over time. If not defined, will interpolate linearly.

fadeEvents *FadeGroup[]*

A list of light fade events to perform in sequence.

activateOnComplete *GO*

Optional object to activate upon completion of all Fade Events.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of all Fade Events. If false, this component will be disabled instead.

Material Fader

Tool: at_MaterialFader

Description:

Transcripts the color on a given object's material to a specified color over time.

Purpose:

To fade a given object to a particular color, or to fade it in or out by fading the alpha channel.

Properties:

FadeChannel *enum*

All	Fade red, green, blue and alpha together.
Alpha	Fade only the alpha channel.
Colors	Fade red, green and blue together.
RedOnly	Fade only the red channel.
GreenOnly	Fade only the green channel.
BlueOnly	Fade only the blue channel.

FadeDetail *class*

targetLabel *string*

Optional name of this material fade. This will overwrite the generic 'Element' default label.

materialSlot *int*

The material slot to fade.

fadeChannel *FadeChannel*

The channels of color to fade.

fadeColor *Color*

The color to fade to.

fadeDelay *float*

Time in seconds from activation of this tool until the fade begins.

fadeDuration *float*

The duration of the fade in seconds.

fadeInterpolation *AnimationCurve*

Optional rate of fade over time. If not defined, will interpolate linearly.

fadeObject *GO*

The object whose Mesh Renderer or Skinned Mesh Renderer will fade. If none is specified, this game object will be used as the Fade Object.

fadeTargets *FadeDetail[]*

A list of material targets to fade between in sequence.

activateOnComplete *GO*

Optional object to activate upon completion of all Fade Targets.

resetOnCompelte *bool*

If true, will deactivate this game object upon completion of all Fade Targets. If false, this component will be disabled instead.

Material Trigger

Tool: at_MaterialTrigger

Description:

A material switcher to override prefab/mesh texture configurations at runtime. Configurable mesh, material and material slot reference.

Purpose:

To specify arbitrary material configuration on prefab characters or miscellaneous game objects.

Properties:

MaterialSwap *class*

swapLabel *string*

Optional name of this material swap. This will overwrite the generic 'Element' default label.

swapObject *GO*

The object whose material is to be swapped when this tool is activated.

swapAllChildren *GO*

If true, will perform this Material Swap on all of the Swap Object children instead.

swapMaterial *Material*

The material to swap to.

materialSlot *int*

The index to swap within the Materials list found on the renderer of the Swap Object. Like all arrays, the first element has an index of zero.

matSwaps *MaterialSwap[]*

A list of material swaps to perform.

resetOnComplete *bool*

If true, will deactivate this game object upon material swap and respect the swap upon re-activation. If false, this component will be disabled instead.

Material Warper

Tool: at_MaterialWarper

Description:

Animates the texture of a material in a cycle over a specified time, in tile scale or offset position.

Purpose:

To move the texture of a material in texture space over time, and make static textures appear fluid or otherwise dynamic.

Properties:

warpTileX *AnimationCurve*

The change in X scale value of the tiling for the material over time.

warpTileY *AnimationCurve*

The change in Y scale value of the tiling for the material over time.

warpTilingCycle *float*

The time in seconds is takes to loop through the values for warp tile scale.

warpOffsetX *AnimationCurve*

The change in X position value of the material, as an offset of the source texture, over time.

warpOffsetY *AnimationCurve*

The change in Y position value of the material, as an offset of the source texture, over time.

warpOffsetCycle *float*

The time in seconds it takes to loop through the values for warp offset position.

randStartOffset *bool*

If true, the tile and offset values will be randomized at scene start. If false, they will start at their beginning curve value.

Mover Series

Tool: at_MoverSeries

Description:

Will move a game object to a series of target game object locations, orientations and/or scales in sequence. Options include delay, duration, separate ease in and ease out, lock rotation Z at zero, use shortest rotation, game object activation upon completion, and smoothing out series movement.

Purpose:

To move an object in the scene from a set position, orientation and/or scale through those defined by multiple target game objects.

Properties:

SimpleMove *class*

moveLabel *string*

Optional name for this move in the series. This will overwrite the generic 'Element' default label.

moveTarget *GO*

The object whose transform represents the target position, rotation and scale of the move.

moveDelay *float*

Time in seconds from activation of this part of the sequence until the move begins.

moveDuration *float*

The duration of the move in seconds.

easeOut *bool*

If true, the move will begin slowly.

easeIn *bool*

If true, the move will end slowly.

useShortestRot *bool*

If true, will rotate towards the target orientation in the shortest direction. If false, rotate by interpolating axis degree values even if that is longer. (Default: true)

zeroRotationZ *bool*

If true, will keep rotation locked at zero along the Z axis. This prevents roll in a camera move, for example. (Default: true)

moveObject *GO*

The game object to move. If not defined, this game object will be moved.

sequence *SimpleMove[]*

A list of moves to perform in sequence.

smoothPath *bool*

If true, will smooth out corners in the translation path for this sequence. Move Targets between the start and end of the sequence can be missed in this mode, as they only influence the path, rather than define positional targets.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

activateOnComplete *GO*

Optional object to activate upon completion of this sequence.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the move series and replay the series upon re-activation.

Mover Single

<u>Tool</u>: at_MoverSingle

<u>Description</u>:

Will move a game object to a target game object position, orientation and/or scale. Configurable properties include delay, move time, ease in/out option, lock Z rotation at zero, use shortest rotation, continuous movement, and reset on complete.

<u>Purpose</u>:

To move an object in the scene from a set position, orientation and/or scale to those defined by a target game object.

<u>Properties</u>:

moveObject *GO*

The game object to move. If not defined, this game object will be moved.

moveTarget *GO*

The object whose transform represents the target position, rotation and scale of the move.

moveDelay *float*

Time in seconds from activation of this tool until the move begins.

moveDuration *float*

The duration of the move in seconds.

easeOut *bool*

If true, the move will begin slowly.

easeIn *bool*

If true, the move will end slowly.

shortestRotation *bool*

If true, will rotate towards the target orientation in the shortest direction. If false, rotate by interpolating axis degree values even if that is longer.

zeroRotationZ *bool*

If true, will keep rotation locked at zero along the Z axis. This prevents roll in a camera move, for example.

continuousMove *bool*

If true, will repeat the move cycle, including delay, and will accumulate all transforms.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the move and replay the move upon re-activation.

Object Tether

Tool: at_ObjectTether

Description:

Define a distance between two game objects, and have the second pulled toward the first, if outside that distance. Configurations for tension.

Purpose:

To tether the one object to another, and have it pulled as if by an invisible rope.

Properties:

tetherHitch *GO*

The object whose transform represents the pulling end of this tether.

tetheredObject *GO*

The object whose transform will be pulled by the Tether Hitch.

tetherLength *float*

The length of allowable slack between the Tether Hitch and the Tethered Object, expressed as distance. (Default: 5.0)

tetherTension *float*

The tension of the tether that translates to the apparent stiffness of the tether. This value represents how aggressively the Tethered Object will be maintained within Tether Length. Typically, this is between zero and one. (Default: 0.1)

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

Parent Trigger

Tool: at_ParentTrigger

Description:

Parents one object to another, with option to only parent and not inherit the parent's transform position and rotation.

Purpose:

To modify the hierarchy and parent / child relationship for purposes of transform inheritance, and/or to snap one object's position or orientation to another.

Properties:

targetGO *GO*

The object whose transform will be parented to, and potentially snapped to match, the Target Parent.

targetParent *GO*

The object whose transform represents the target for parenting or snapping. If this is null, the Target GO will be un-parented.

parentDelay *float*

Time in seconds from activation of this tool until parenting operation occurs.

snapPosition *bool*

If true, move the Target GO to match the position of the Target Parent. This is the same as teleporting the Target GO.

snapRotation *bool*

If true, rotate the Target GO to match the orientation of the Target Parent.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnTrigger *bool*

If true, will deactivate this game object upon parenting and repeat the parenting upon re-activation. If false, will disable this component instead.

Particle Trigger

Tool: at_ParticleTrigger

Description:

A trigger for particle system objects that can play once, play loop or stop its emission.

Purpose:

To control a particle system emission while maintaining the active particles, even if stopped.

Properties:

ParticleAction *enum*

Play	The particle emission will launch new particles as configured, setting Loop to false.
Loop	The particle emission will launch new particles as configured, setting Loop to true.
Stop	The particle emission will stop, but existing particles will continue their configured life.

particleSystemObject *ParticleSystem*

The particle system to trigger and act upon.

particleAction *ParticleAction*

The action to perform on the Particle System Object.

actionDelay *float*

The amount of time in seconds between the activation of this trigger and the action performed.

resetOnTrigger *bool*

If true, this tool with be deactivated upon completion of the action. If false, this tool with be disabled instead.

Render Trigger

Tool: at_RenderTrigger

Description:

Will transition the configured skybox and other render settings to those specified by this tool.

Purpose:

To fade the skybox, fog and ambient lighting to arbitrary settings at runtime. *Note: This is not intended for use with procedural skyboxes.*

Properties:

fogSpecs *class*

fogColor *Color*

The target color of the fog.

fogMode *FogMode*

The Fog Mode to use for the transition. (Default: FogMode.Linear)

fogDensityExp *float*

The target value for the density of exponential fog.

fogNearLinear *float*

The target value for linear fog near distance.

fogFarLinear *float*

The target value for linear fog far distance.

changeSkybox *bool*

If true, will blend the current skybox to Skybox Material.

skyboxMaterial *Material*

The material to blend for the skybox.

changeAmbientLight *bool*

If true, will blend the ambient light from the current color to Ambient Light Color.

ambientLightColor *Color*

The color to blend to the ambient light.

changeFog *bool*

If true, will blend the fog values from the current to the Fog specifications.

fog *fogSpecs*

Target fog settings for the blend.

changeDelay *float*

Time in seconds from activation of this tool until the render settings blend will begin.

changeDuration *float*

The duration of the render settings blend.

Spawn Trigger

Tool: at_SpawnTrigger

Description:

Will create copies of a given prefab at a target game object.

Purpose:

A prefab copier (or factory) to make objects appear in a scene at runtime upon trigger.

Properties:

spawnPrefab *GO*

The prefab object to spawn, as a clone of the prefab reference. This object must be found in a project folder called Resources to be accessible at runtime.

spawnTarget *GO*

The object whose transform position, rotation and scale represent the target transform values for the prefab to be spawned.

spawnName *string*

The name to be given to spawned objects. If none is provided, the Spawn Prefab name will be used with the suffix, "_spawn".

lifespan *float*

The time in seconds in which the spawned object will exist in the scene. If zero, the object will be allowed to exist indefinitely.

maxAlive *int*

The maximum number of spawned objects this factory will allow. If more are created, the oldest will be destroyed. (Default: 1)

resetOnTrigger *bool*

If true, will deactivate this game object upon spawning and repeat the spawning upon re-activation. If false, will disable this component instead.

Sprite Animator

Tool: at_SpriteAnimator

Description:

Will display a series of Sprites in sequence on a given Sprite Renderer object.

Purpose:

To play sprite animations. Options include setting frame rate and to loop sequence.

Properties:

animInterval *float*

The time in seconds between frames of the sprite animation. (Default: 0.2 seconds)

animSprites *Sprite[]*

A list of Sprites to be displayed in sequence.

spriteRenderer *SpriteRenderer*

The Sprite Renderer to display this animation. This renderer should be set to the first Sprite of the sequence.

loop *bool*

If true will continue to repeat this animation sequence from the beginning. If false, will stop once the sequence is complete.

activateOnComplete *GO*

Optional object to activate upon completing the sequence.

resetOnComplete *bool*

If true, will deactivate this game object upon completing the sequence and replay the sequence upon reactivation. If false, this component will be disabled instead. However, if Loop is true, this control will be ignored.

Teleport Trigger

Tool: at_TeleportTrigger

Description:

Will teleport a game object, namely the player avatar, to a specified location.

Purpose:

To teleport objects to an arbitrary position and orientation.

Properties:

teleportObject *GO*

The object to be teleported.

teleportTargetObject *GO*

The object whose transform represents the position and potentially the rotation for the teleport event.

teleportDelay *float*

Time in seconds from activation of this tool until the teleport event occurs.

ignoreRotation *bool*

If true, the Teleport Object will not be rotated to match the orientation of the Teleport Target Object.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnTrigger *bool*

If true, will deactivate this game object upon teleport and repeat the teleport event upon re-activation. If false, will disable this component instead.

Texture Animator

Tool: at_TextureAnimator

Description:

Will play a series of Textures in sequence on a given Material.

Purpose:

To play texture animations. Options include setting frame rate and to loop sequence.

Properties:

animInterval *float*

The time in seconds between frames of the texture animation. (Default: 0.2 seconds)

animTextures *Texture[]*

A list of Textures to be displayed in sequence.

animMaterial *Material*

The Material to display this animation. The base texture for this Material should be set to the first Texture of the sequence.

loop *bool*

If true will continue to repeat this animation sequence from the beginning. If false, will stop once the sequence is complete.

activateOnComplete *GO*

Optional object to activate upon completing the sequence.

resetOnComplete *bool*

If true, will deactivate this game object upon completing the sequence and replay the sequence upon reactivation. If false, this component will be disabled instead. However, if Loop is true, this control will be ignored.

Time Dilator

Tool: at_TimeDilator

Description:

Temporarily adjusts the scale at which time progresses.

Purpose:

To present slow motion or fast motion effects at runtime.

Properties:

timeDilation *float*

The rate at which time till appear to pass, expressed as a number that is more than zero, where standard time is equal to one.

timeDilationDelay *float*

Time in seconds from activation of this tool until time is dilated.

timeDilationDuration *float*

The duration of the time dilation event in seconds. The duration value will represent real world time if Dilate Duration below is true.

dilateDuration *bool*

If true, the duration specified above will be automatically scaled by Time Dilation at runtime, so the Dilation Duration value as configured can represent time in real world seconds. If false, the duration of the Time Dilation event itself will be dilated. (Default: true)

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the time dilation event and repeat the event upon re-activation. If false, will disable this component instead.

Volume Thruster

Tool: at_VolumeThruster

Description:

Applies thrust within this game object’s collision volume, based on given world space vector and magnitude, to valid colliding game objects.

Purpose:

To push, nudge or launch physics-enabled game objects, such as character controllers. Note: This game object must have a collider component set to isTrigger = true

Properties:

thrustVector *Vector3*

The direction of thrust to apply to valid Rigidbody colliders or character controllers that fall within this tool's Collider. If this is zero, will radiate thrust from the center of this object. (Default: Vector3.up)

useLocalSpace *bool*

If true, will orient the thrust vector based on the local rotation of this game object. If false, the thrust vector will be based on global space.

thrustMagnitude *float*

The amount of thrust to be applied in the direction of the Thrust Vector or radially if zero, expressed as distance per frame.

validCollider *GO*

Optional reference to a specific object to allow, to the exclusion of all other colliders. If this is null, any Rigidbody collider or character controller will be considered valid. Note Rigidbody and character motor components handle gravity in independent ways and differ by default settings.

Audio Fader

Tool: at_AudioFader

Description:

Will transition the volume of audio sources on a game object to a specified volume.

Purpose:

To fade volumes of audio objects up or down.

Properties:

audioObject *GO*

The object whose Audio Source volume values will be faded. The volume will fade from a single volume value, the first one found. Typically, audio objects should therefore be grouped if they are intended to share the same volume value when this Fade Event begins.

fadeAllChildren *bool*

If true, will fade all found Audio Sources on children of the Audio Object. If false, will simply adjust Audio Sources on the Audio Object.

targetVolume *float*

The volume value to fade to. This must be a value between zero and one.

fadeDelay *float*

Time in seconds from activation of this tool until the fade event occurs.

fadeDuration *float*

The duration of the fade in seconds.

fadeInterpolation *AnimationCurve*

Optional rate of fade over time. If not defined, will interpolate linearly.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the fade event and repeat the event upon re-activation. If false, will disable this component instead.

Audio Series

Tool: at_AudioSeries

Description:

Plays a series of audio objects (game objects with Audio Source components), with optional time padding at start and in-between. Options to activate an object at the end of the series, and to parent the audio object to another game object (like the player avatar).

Purpose:

To play a series of audio objects, automatically timed to the length of each clip. A simple way to present dialog in cinematic sequences.

Properties:

Step *class*

stepLabel *string*

Optional name for this step. This will overwrite the generic 'Element' default label.

audioSequence *GO[]*

A list of audio objects to activate in sequence. An audio object is simply a game object with an Audio Source component.

timePadding *float*

Time in seconds between activation of each audio object in the Audio Sequence. Note all audio objects will be deactivated upon activation of this tool, if they are not already.

activateOnStep *GO*

Optional object to activate upon reaching this Audio Step.

audioParent *GO*

Optional object to parent audio objects to, and define position for, when first activated.

audioSteps *Step[]*

A list of audio objects to play in sequence.

activateOnComplete *GO*

Optional object to activate upon completion of this audio series.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the series and repeat the series upon re-activation. If false, will disable this component instead.

Audio Single

Tool: at_AudioSingle

Description:

Delays the start of an audio play and/or delays the loop of an audio source on this game object. Options to specify range of random loop delay, and to specify a range of variation in pitch.

Purpose:

To have an audio source offset in time, or to make a looping audio source vary each loop in pitch or time. In other words, to make a looping audio clip not immediately recognizable as looping.

Properties:

startDelay *float*

Time in seconds from activation of this tool until audio play.

loopDelay *float*

If the Audio Source component Loop property is true, this value is the time in seconds between plays of the audio clip.

maxLoopDelay *float*

Optional maximum range of loop delay in seconds, to be chosen at random.

pitchVariance *float*

Variation in pitch between loop audio plays, expressed as a percentage.

resetOnTrigger *bool*

If true, will deactivate this game object upon playing once and respect Start Delay and Pitch Variance upon re-activation. If false, will disable this component instead unless Audio Source Loop is true.

Music Manager

Tool: at_MusicManager

Description:

Will handle transitions between music audio clips, as triggered by a Music Trigger. Option for transition speed.

Purpose:

Manage music in scenes, so as to cross-fade between music tracks.

Properties:

fader *float*

The current position of the cross fader between two Audio Sources created here at runtime, which are used to mix music audio over time.

faderTarget *float*

The target position of the fader. This is meant to be a number between zero and one, representing a percentage from the first Audio Source to the second.

faderSpeed *float*

The relative rate at which the fader will transition between audio sources, expressed as a percentage per frame. (Default: 0.05, or 5%)

Music Trigger

Tool: at_MusicTrigger

Description:

Calls on a Music Manager to load a music audio clip and transition to it via cross fade.

Purpose:

To trigger the next music track to play or to fade out music.

Properties:

musicClip *AudioClip*

The Audio Clip to play on the Music Manager. This will be mixed into any current music audio via cross fade, and if this is null, any current music will fade out. If this clip matches the current music audio, the trigger will be ignored rather than cross-fading into a duplicate clip. The Audio Clip must be in a folder named Resources to be available for loading at runtime.

relayDelay *float*

Time in seconds from activation of this tool until music mix begins.

resetOnTrigger *bool*

If true, will deactivate this object upon music mix and repeat the mix upon re-activation. If false, will disable this component instead.

SFX Object

Tool: at_SFXObject

Description:

Identifies this game object as an audio object that can be muted via Menu Manager. Option includes defining the type of audio, matching options for toggling as defined from the Scene Manager.

Purpose:

To make sure audio objects are muted when the appropriate option is toggled off.

Properties:

SoundType *enum*

SFX	This is a sound effect audio object.
Music	This is a musical ‘sting’ (non-looping clip) audio object.
Voice	This is a voice audio object.
Interface	This is an interface audio object.
Movie	This is a movie audio object.

soundType *SoundType*

The type of sound this audio object represents for mute controls as defined by the Scene Manager. Note that Music is available for musical 'stings', as opposed to looping music tracks, handled by the Music Manager.

Camera Manager

Tool: at_CameraManager

Description:

Turns camera objects on/off in sequence, with target camera object references. Options for delay & duration of cut, use of separate microphone object, flow control and game object to activate upon cut completion.

Purpose:

To present a cinematic sequence, a series of camera shots that cut from one camera to another.

Properties:

managedCamera *Camera*

The camera that will be suspended during this cinematic sequence. Typically this is the Main Camera object, and if none is specified, this value will default to Main Camera at runtime.

CineCutCam *class*

cutLabel *string*

Optional name for this shot. This will overwrite the generic 'Element' default label.

cutToCam *Camera*

The camera used in this shot. Camera objects will be deactivated at the beginning of this cinematic and activated in sequence.

cutDuration *float*

The duration of the shot in seconds. However, if this value is zero, the shot length will be considered infinite.

activateOnCut *GO*

Optional reference to an object to activate upon reaching this shot.

cineCuts *CineCutCam[]*

A list of shots in this cinematic sequence.

useMicrophone *bool*

If true, Audio Listeners will be disabled and use of a separate Audio Listener tool is assumed. If false, Audio Listeners will be toggled on each camera object in sequence.

activateOnComplete *GO*

Optional reference to an object to activate upon completion of this cinematic sequence.

Camera Shaker

Tool: at_CameraShaker

Description:

Shakes a specified camera with multiple configurations available.

Purpose:

To cause a camera to shake, as if bumped, rattled, rocked or hand-held, depending on settings given.

Properties:

targetCamera *Camera*

The camera to shake. If not specified, at runtime this will default to a Camera component on this object, or if nothing else, the Main Camera.

shakeDelay *float*

Time in seconds from activation of this tool until camera shake begins. (Default: 0.1 seconds)

shakeDuration *float*

The duration of the shake in seconds. (Default: 1 second)

shakeAttack *float*

A measure of how forceful the shake performs initially, a number between zero and one. The higher the number, the more abrupt the shake will begin.

shakeFalloff *float*

The amount the shake should ramp down at the end, a number between zero and one. The higher the number, the more gradual the shake will fade out. (Default: 1.0)

shakeCamPos *Vector3*

The amplitude of shake position along each axis of movement, expressed as distance.

posVariance *float*

The amount of variation applied randomly to position shake along each axis, expressed as distance. (Default: 0.1 units)

shakeCamRot *Vector3*

The amplitude of shake rotation along each axis of rotation, expressed in degrees.

rotVariance *float*

The amount of variation applied randomly to rotation shake along each axis, expressed as degrees. (Default: 1 degree)

shakeFrequency *float*

The number of shake cycles to perform per second.

frequencyVariance *float*

The amount of variation applied randomly to frequency each frame, expressed as shake cycles per second. Note this variance has a lot of control over shake results. (Default: 0.1)

returnOnComplete *bool*

If true, will return the Target Camera to the transform position and rotation found at the beginning of the shake. If false, will leave the Target Camera at arbitrary transform values at the end of the shake.

resetOnCompelte *bool*

If true, will deactivate this game object upon completion of the shake event and repeat the event upon re-activation. If false, will disable this component instead.

Camera Zoomer

Tool: at_CameraZoomer

Description:

Interpolates between current and target field of view values on a camera.

Purpose:

To present a zoom in/out effect on cameras.

Properties:

zoomCamera *Camera*

The camera to zoom. If none is defined, will default to the camera component on this object, or the main camera if that fails.

targetFieldOfView *float*

The Field Of View target value to zoom toward. This must be a value between one and 179.

zoomDelay *float*

Time in seconds from activation of this tool until camera zoom begins.

zoomDuration *float*

The duration of the zoom in seconds.

zoomInterpolation *AnimationCurve*

Optional rate of zoom over time. If not defined, will interpolate linearly.

resetOnComplete *bool*

If true, will deactivate this tool upon completion of the zoom and repeat the zoom upon re-activation.

Captions Manager

Tool: at_CaptionsManager

Description:

Will manage a series of text captions at the bottom of the screen when called by a Captions Trigger.

Purpose:

To present text captions or dialog without recording vocal audio.

Properties:

Caption *class*

captionSpeaker *string*

Optional name of the speaker, to be formatted as '*Speaker: Text*'.

captionText *string*

The caption text to display.

captionDuration *float*

The duration of this caption in seconds.

captions *Caption[]*

A list of screen captions to display in sequence.

captionsPadding *float*

Time in seconds between display of each caption. (Default: 0.25 seconds)

captionsDepth *int*

The layer order of the captions. The larger the number, the more behind the layer is configured to display the caption text. (Default: -1)

captionsMuted *bool*

If true, will hide captions. Note that captions are not paused if muted.

captionsFontSize *int*

The font size of the captions text as displayed at a screen width of 1024 pixels. The font will scale appropriately. This value will override the font size as defined in its GUI Style. (Default: 30)

captionsTextColor *Color*

The top color of the captions text. This value will override the normal text color as defined in its GUI Style. (Default: Color.white)

captionsTextPadding *float*

The amount of padding around the captions text. This is measure in proportions of screen space; typically zero to 0.5. This value will override the padding as defined in its GUI Style. (Default: 0.05, or 5% of screen space)

captionsStyle *GUIStyle*

An optional style for the captions display, to change the font of the captions text, for example.

Captions Trigger

Tool: at_CaptionsTrigger

Description:

Will call the Caption Manager in the scene and load specified captions to play in sequence.

Purpose:

To trigger the next captions to display.

Properties:

captions *at_CaptionsManager.Caption[]*

The screen captions to send to the captions manager for display.

captionsDelay *float*

Time in seconds from activation of this tool until the captions are sent to the captions manager.

useAutoDurations *bool*

If this is true, Caption Durations will be automatically calculated at runtime, as an approximation based on the length of the Caption Text. If false, Caption Duration will be used as configured.

resetOnTrigger *bool*

If true, will deactivate upon trigger and respect CaptionsDelay upon re-activation.

Movie Trigger

Tool: at_MovieTrigger

Description:

Will control standard .ogg / .ogv format movie file playback.

Purpose:

To play, pause, stop or replay movies upon trigger.

Properties:

MovieAction *enum*

Play	Play the movie and audio from the current position.
Pause	Pause the movie and audio.
Stop	Stop the movie and audio, and rewind the current position to the beginning.
Replay	Rewind the movie and audio from the beginning and play them.

movieScreen *GO*

The object whose Renderer is to display the movie. This Renderer's Material main texture must be of type MovieTexture to be valid. This can be created by dragging the movie reference onto the object.

movieAction *MovieAction*

The movie action to trigger. Note that Stop will both stop and rewind the movie. (Default: MovieAction.Play)

movieSpeaker *GO*

An optional object whose Audio Source is to play the audio with the movie. If not specified, at runtime this will default to the Movie Screen object and an Audio Source will be added automatically.

resetOnTrigger *bool*

If true, will deactivate this game object upon completion of the Movie Action and repeat the action upon re-activation. If false, will disable this component instead.

Overlay Fader

Tool: at_OverlayFader

Description:

Manages images meant to overlay the viewport. Options for fade up and down, with delay and duration.

Purpose:

To present title cards or to fade to/from black, or to make use of faux displays for various visual effects (such as the Half Life 2 menu 'unfocused' effect)

Properties:

overlayImage *Texture*

The texture swatch or image to display over the entire screen.

overlayDepth *int*

The layer order of this fader. The larger the number, the more behind the layer is configured to display the overlay.

overlayFadeUp *bool*

If true, will fade this Overlay Image up from clear.

fadeUpStart *float*

Time in seconds from activation of this tool until fade up begins.

fadeUpDuration *float*

The duration of the fade up in seconds.

fadeUpInterpolation *AnimationCurve*

Optional rate of fade up over time. If not defined, will interpolate linearly.

overlayFadeDown *bool*

If true, will fade this Overlay image down to clear.

fadeDownStart *float*

Time in seconds from the end of the fade up until fade down begins. The Fade Up Start and Fade Up Duration time values will be respected even if Overlay Fade Up is false.

fadeDownDuration *float*

The duration of the fade down in seconds.

fadeDownInterpolation *AnimationCurve*

Optional rate of fade down over time. If not defined, will interpolate linearly.

resetOnComplete *bool*

If true, will respect all fade timing upon reactivation of this tool. If false, this tool will be disabled upon completion of all fades.

Cheat Manager

Tool: at_CheatManager

Description:

Will activate, deactivate or toggle another game object upon detection of a given string of alphanumeric characters.

Purpose:

To trigger objects based on entered cheat codes.

Properties:

CheatAction *enum*

Activate	The trigger object will be activated when this code is detected.
Deactivate	The trigger object will be deactivated when this code is detected.
Toggle	The trigger object will be toggled when this code is detected.

CheatEvent *class*

cheatCode *string*

The unique string of alphanumeric characters to check. Each of these characters must be entered in sequence to be a valid cheat code.

objectToTrigger *GO*

The object to act upon when the Cheat Code is detected. The object will be activated or deactivated or toggled.

cheatAction *CheatAction*

The action to take once the Cheat String is detected.

codes *CheatEvent[]*

The cheat codes to act upon. To avoid confusion, make sure no cheat code sequence can be found within another.

Counter Trigger

Tool: at_CounterTrigger

Description:

Will activate another game object upon being triggered a specified count number of times.

Purpose:

To handle accumulative triggers that will then activate another game object.

Properties:

targetCount *int*

The number to count to before activating another object. (Default: 1)

countOnActivate *bool*

If true, will count based on activation of this object. If false, will count based on this Collider's trigger event.

activateOnComplete *GO*

The object to activate upon reaching the Target Count.

validCollider *GO*

Optional reference to a specific object to allow during this Collider's trigger event. If this is null, any capsule collider or character controller will be considered valid.

configureFor2D *bool*

If true, 2D Collider components will be used as a basis for collision trigger. If false, 3D Collider components will be used instead.

Distance Measure

Tool: at_DistanceMeasure

Description:

A tool that calculates the distance value between two game objects.

Purpose:

To measure the distance between objects and store the result for other tools, such as Math Relay.

Properties:

startMeasure *GO*

One of two game objects to measure distance between.

endMeasure *GO*

The other game object to measure distance between.

Distance *float*

The measured distance between Start Measure and End Measure objects each frame. This property is automatically updated at runtime.

Economy Manager

Tool: at_EconomyManager

Description:

Will keep track of an arbitrary resource amount, receive trigger input to adjust that amount, and trigger other game objects when given thresholds are reached. Can display the amount as a HUD item at given screen locations.

Purpose:

To track and display various economies, and to serve as general managers of the resource.

Properties:

EconType *enum*

IntegerNum	This economy is based on integers, or whole numbers.
FloatNum	This economy is based on floating-point numbers, or decimal numbers.
PercentNum	This economy is based on a percentage, displayed with a "%".
TimeSec	This economy is time-based, displaying seconds.
TimeMinSec	This economy is time-based, displaying minutes and seconds, as MM:SS.
TimeMinSecHun	This economy is time-based, displaying minutes, seconds and hundredths of a second, as MM:SS.HH.

MarkAction *enum*

Activate	The trigger object will be activated when this value is reached.
Deactivate	The trigger object will be deactivated when this value is reached.
Toggle	The trigger object will be toggled when this value is reached.

EconPoint *class*

markLabel *string*

Optional name for this Economy Mark. This overwrites the default 'Element 0' label.

markValue *float*

The economy value to detect and act upon.

onMarkTrigger *GO*

The object to act upon when the Mark Value is reached. The object will be activated or deactivated or toggled.

onMarkAction *MarkAction*

The action to take once the Mark Value is reached.

economyType *EconType*

The type of economy this manager is to track. This type determines values tracked as well as display. If a Time type, will act as a timer when triggered.

economyMin *float*

The minimum value the economy can reach.

economyMax *float*

The maximum value the economy can reach.

economyStart *float*

The starting value of the economy.

economyRoundTo *int*

The decimal place to round the economy value. If the Economy Type is IntegerNum, this property is ignored.

economyMarks *EconPoint[]*

A list of values of this economy, that when reached, various actions may be performed as a result.

economyDisplay *string*

The text to display for this economy. This value is automatically updated at runtime.

displayEconomy *bool*

If true, will display the economy value in the viewport. (Default: true)

displayTop *float*

The top edge of the economy value display, expressed as a ratio of vertical screen space, which is typically a number between zero and one.

displayLeft *float*

The left edge of the economy value display, expressed as a ratio of horizontal screen space, which is typically a number between zero and one.

displayFontSize *int*

The font size of the economy value display. This is the font size as displayed at a screen width of 1024 pixels, and will be scaled appropriately. (Default: 20)

displayColor *Color*

The color of the economy value display. (Default: Color.white)

displayDepth *int*

The layer order of the economy value display. The larger the number, the more behind the layer is configured to display the economy value.

Economy Trigger

Tool: at_EconomyTrigger

Description:

Will change the value of an economy by a given amount.

Purpose:

To trigger economy changes. Options for incrementing, decrementing, and amount.

Properties:

economyManager *at_EconomyManager*

The Economy Manager to trigger.

triggerAmount *float*

The amount to change the managed economy value. This can be positive or negative number. If a time-based economy, any positive or negative value will begin the timer to run, while a zero value will cause the timer to stop.

relayOnActivate *bool*

If true, trigger the change in economy based on time from activation of this tool. If false, trigger based on this Collider's trigger event.

relayDelay *float*

The number of seconds between the activation of this tool and the change in economy value.

validCollider *GO*

Optional reference to a specific object to allow during this Collider's trigger event. If this is null, any capsule collider or character controller will be considered valid.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnTrigger *bool*

If true, will deactivate this tool upon trigger and repeat the trigger upon re-activation. If false, will disable this component instead.

configureFor2D *bool*

If true, this tool will work with only 2D collider components. If false, this tool will work with only 3D collider components. This setting is only necessary if Relay On Activate is false.

Event Manager

Tool: at_EventManager

Description:

Activates and/or deactivates objects as a group, and in non-sequential steps. Option available to only enable and/or disable the script components found on these objects; keeping the objects, and any child objects, active.

Purpose:

To toggle preset game objects in scene at runtime, and to cause general state changes in scene by use of an Event Trigger.

Properties:

Step *class*

eventLabel *string*

Optional name for this event. This will overwrite the generic 'Element' default label.

objectsToActivate *GO[]*

A list of objects to activate upon trigger of this event.

objectsToDeactivate *GO[]*

A list of objects to deactivate upon trigger of this event.

affectScriptsOnly *bool*

If true, will enable or disable all MonoBehaviour script components found on event reference objects and leave the activation state of the objects as they are. If false, will simply activate or deactivate event reference objects, which has a recursive effect on any of their child objects.

eventSteps *Step[]*

Each event that can be triggered, as expressed by objects that are activated or deactivated.

Event Series

Tool: at_EventSeries

Description:

Will act as a sequence of Event Triggers upon a given Event Manager.

Purpose:

To play out a set number of events in a timed sequence.

Properties:

eStep *class*

eventStep *int*

The Event Step index to request from the Event Manager when this event is triggered. Like all arrays, the first element has an index of zero.

eventDelay *float*

Time in seconds from reaching this step in the sequence until the event is triggered and the next step is reached.

eventManager *GO*

The Event Manager object to call when this event is triggered.

eventSequence *eStep[]*

A list of event steps to trigger in sequence.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of the sequence and repeat the sequence upon re-activation. If false, will disable this component instead..

Event Single

Tool: at_EventSingle

Description:

Will call an Event Manager game object to process an event step. Options include delay, a valid collider reference and a conditional object that must be active to allow the event to trigger.

Purpose:

To trigger event steps to change, either by collision trigger or by timed event.

Properties:

eventManager *GO*

The Event Manager object to call when this event is triggered.

eventStep *int*

The Event Step index to request when this event is triggered. Like all arrays, the first element has an index of zero.

relayOnActivate *bool*

If true, call the event to be triggered based on time from activation of this tool. If false, call the event based on this Collider's trigger event.

relayDelay *float*

Time in seconds from activation of this tool until the event can be triggered. Note this tool will then continue to attempt to trigger the event if Conditional If Active refers to a currently deactivated object.

validCollider *GO*

Optional reference to a specific object to allow during this Collider's trigger event. If this is null, any capsule collider or character controller will be considered valid.

conditionalIfActive *GO*

Optional reference to an object whose activation state will determine whether the event will be called. The event will only be triggered if and when the object is active.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnTrigger *bool*

If true, will deactivate this game object upon trigger and respect any Relay Delay upon re-activation. If false, will disable this component instead.

configureFor2D *bool*

If true, 2D Collider components will be used as a basis for collision trigger. If false, 3D Collider components will be used instead.

Input Relay

Tool: at_InputRelay

Description:

Toggles the active state of game objects that are associated with given key code strings

Purpose:

For use of arbitrary user input in trigger systems.

Properties:

InputPair *class*

buttonName *string*

The name of the button or key to check. If using virtual buttons, ensure it is configured and valid before using, under Edit -> Project Settings -> Input

objectToActivate *GO*

The object to activate upon detection of the button or key down.

deactivateOnRelease *bool*

If true, will deactivate the object upon detection of the button or key up.

firstPressOnly *bool*

If true, will only consider the first press of the button or key.

useVirtualButtons *bool*

If true, will use the preconfigured Axes found in the Input Manager. Ensure it is configured and valid before using, under Edit -> Project Settings -> Input. (Default: true)

inputs *InputPair[]*

The inputs to check each frame.

Interval Trigger

Tool: at_IntervalTrigger

Description:

Will activate other game objects upon reaching time intervals.

Purpose:

To trigger objects repeatedly, given specified time interval lengths and variance.

Properties:

IntervalMode *enum*

Regular	The target object will be activated at a static interval, based on Interval.
Range	The target object will be activated at a random interval based on Interval and the Interval Variance.
Gaussian	The target object will be activated at a random interval based on Interval and the Interval Variance, where the result will skew to make extreme variance less likely. (i.e., a bell curve distribution)

intervalMode *IntervalMode*

The time Interval Mode of this trigger.

Interval *float*

The base time in seconds for the interval.

intervalVariance *float*

The amount in seconds that the interval can vary, if the Interval Mode is not Regular.

activateOnTrigger *GO*

The object to activate upon reaching each interval.

Logical Trigger

Tool: at_LogicalTrigger

Description:

Will evaluate logical conditions of other game object's activation state and trigger another game object if evaluated as true.

Purpose:

To trigger another object based on complex conditions.

Properties:

LogicalMode *enum*

Not	The target object will be activated if all key objects are not active.
And	The target object will be activated if all key objects are active.
Or	The target object will be activated if any key object is active.
Xor	The target object will be activated if only one key object is active.

logicMode *LogicalMode*

The Logic Mode of this trigger, determining the condition to activate a target object.

logicalKeyObjects *GO[]*

A list of objects whose active state are the conditions to be checked per the Logical Mode.

logicalTarget *GO*

The object to activate if the Logical Mode conditions evaluate as true, per the Logical Mode setting.

deactivateIfFalse *bool*

If true, will deactivate the Logical Target object if the Logical Mode conditions evaluate as false.

resetOnTrigger *bool*

If true, will deactivate upon evaluating and evaluate again once re-activated. If false, will disable this component instead.

Look At Trigger

Tool: at_LookAtTrigger

Description:

Tracks the viewing angle of a camera for the purpose of activating a target object when this game object is 'looked at'; and deactivating when 'not looked at'. Includes configurations for viewing angle tolerance.

Purpose:

To track whether player camera is able to view particular locations.

Properties:

lookerCamera *Camera*

The Camera to track. Another object will be activated when the Looker Camera is looking at the position of this tool.

useAngleTolerance *bool*

If true, will determine whether looked at based on the angle of the Looker Camera to this tool and if that falls within the Angle Tolerance. If false, any appearance of the position of this tool within frame of the Looker Camera is considered looked at.

angleTolerance *float*

The angle in degrees from center of Looker Camera view that is considered looked at if Use Angle Tolerance is true. (Default: 30 degrees)

activateOnTrigger *GO*

The object to activate when this tool is looked at and to deactivate when not.

Math Relay

Tool: at_MathRelay

Description:

Reads a numeric value from a property on a given tool, performs a math operation, and stores the result value. Option to then pass the result to another tool's numeric property.

Purpose:

To perform a math operation and relay the result to another tool.

Properties:

MathOperation *enum*

Addition	Add the source to the Operation Value.
Multiplication	Multiply the source by the Operation Value.
NoMoreThan	Clamp the source to no more than the Operation Value.
NoLessThan	Clamp the source to no less than the Operation Value.
RemainderAfterDividedBy	Divide the source by the Operation Value and provide the remainder.

scriptSource *MonoBehaviour*

The MonoBeahviour tool that contains the source property to operate on.

propertySource *string*

The float or integer property name to use as the source value for math operation.

operation *MathOperation*

The type of math operation to perform on the source property value.

operationValue *float*

The value to use in the math operation.

resultValue *float*

The math operation result value. This property will be automatically updated upon math relay.

scriptDestination *MonoBehaviour*

Optional MonoBehaviour tool that contains the destination property to relay the math result.

propertyDestination *string*

Optional float or integer property name to use as the destination of the math result.

activateOnComplete *GO*

Optional object to activate upon completion of operation.

resetOnComplete *bool*

If true, will deactivate this game object upon completion of operation. If false, will disable this component instead..

Random Trigger

Tool: at_RandomTrigger

Description:

Selects randomly from an array of game object references to activate the selection. Options for random selection mode, deactivating others not selected and triggering a game object after the full array has been selected. (RandomExhaustive mode only)

Purpose:

To randomly activate, or trigger, one object from a list of objects.

Properties:

RandMode *enum*

RandomRepeat	Select to activate any target object, regardless of repetition.
RandomNoRepeat	Select to activate any target object, except the last selected object.
RandomExhaustive	Select to activate any target object, unless it has already been selected.

targetArray *GO[]*

A list of objects to select for activation at random.

selectionMode *RandMode*

How selections are made with respect to repetition and whether all options should be exhausted before repeating any. (Default: RandMode.RandomExhaustive)

deactivateUnselected *bool*

If true, will deactivate all objects in the Target Array that are not selected.

triggerOnComplete *GO*

Optional object to activate upon exhausting all selections in the Target Array, if the Selection Mode is Random Exhaustive.

resetOnTrigger *bool*

If true, will deactivate this game object upon trigger and re-select randomly upon re-activation. If false, will disable this component instead.

Script Trigger

Tool: at_ScriptTrigger

Description:

Will call arbitrary methods on provided game object references based on the given message string. Options include delay, a valid collider reference and values to pass to the script methods.

Purpose:

A trigger to use Unity's SendMessage() to call arbitrary script functionality on game objects.

Properties:

ValueType *enum*

None	The Value To Pass is intended to be empty. There is no value send.
Boolean	The Value To Pass is a Boolean value, or value of True or False.
Integer	The Value To Pass is an integer value, or a whole number.
Float	The Value To Pass is a floating point value, or decimal number.
String	The Value To Pass is a string value, or an alphanumeric text value.

receiverObject *GO*

The object whose script component is to be called. Note that this call will broadcast to every script component on this object.

methodToCall *string*

The method or function to call on the Receiver Object. This method must be public, and it may take one or zero arguments below as Value To Pass. This string is the name of the method, without parenthesis.

valueToPass *string*

Optional method value to send with the call as an argument. This allows the script method to receive data with the call event. Be sure to set an appropriate Value Type.

valueType *ValueType*

The type of data represented with Value To Pass, if any.

relayOnActivate *bool*

If true, call the event to be triggered based on time from activation of this tool. If false, call the event based on this Collider's trigger event.

relayDelay *float*

Time in seconds from activation of this tool until the method is called.

validCollider *GO*

Optional reference to a specific object to allow during this Collider's trigger event. If this is null, any capsule collider or character controller will be considered valid.

useFixedUpdate *bool*

If true, this tool will update based on a fixed measure of time, used just prior to physics updates on Rigidbody objects. If false, this will be updated each frame, which is variable and can be slightly out of sync with physics.

resetOnTrigger *bool*

If true, will deactivate this game object upon calling the Receiver Object and repeat the call upon re-activation. If false, will disable this component instead.

configureFor2D *bool*

If true, 2D Collider components will be used as a basis for collision trigger. If false, 3D Collider components will be used instead.

Spawn Life

Tool: at_SpawnLife

Description:

Will destroy this game object when a lifespan time duration has ended.

Purpose:

To clean up game objects from a scene. Automatically added to objects by Spawn Triggers.

Properties:

lifespan *float*

The time in seconds this object is allowed to exist. After this time, the object will be destroyed.

spawnTrigger *GO*

Reference to the Spawn Trigger that spawned this object, if any.

Tool Setup

Tool: at_ToolSetup

Description:

Will set properties on tools at runtime with given input. Options include type of input and the ability to find game object or script-type inputs based on unique game object name.

Purpose:

To reconfigure tools at runtime. To set up spawned tools. To set up tools in a scene with tools that migrated from another upon load.

Properties:

PropType *enum*

Boolean	The input is a Boolean value, or value of True or False.
Integer	The input is an integer value, or a whole number.
Float	The input is a floating point value, or decimal number.
String	The input is a string value, or an alphanumeric text value.
GameObject	The input is a Game Object value, referencing a game object.
Script	The input is a script value, referencing a MonoBehaviour script tool.

setupDelay *float*

The time in seconds from the time this tool is activated until the setup is performed.

tool *MonoBehaviour*

The MonoBehavior script tool that will be set up according to given Property Name, Type and relevant Input. Note that changing values of tool properties as runtime may cause unexpected results.

propertyName *string*

The name of the property to be set up. Note that property names will begin with a lower case letter, and there will be no spaces in the name.

propertyType *PropType*

The type of value to be passed to the property as input. The appropriate input field below should be populated. Note that all other input values will be ignored.

boolInput *bool*

If Property Type is Boolean, this value will be passed to the Tool.

intInput *int*

If Property Type is Integer, this value will be passed to the Tool. Note that this will also work with Enum properties, seen as pull down lists just like Property Type, where the first element is zero.

floatInput *float*

If Property Type is Float, this value will be passed to the Tool.

stringInput *string*

If Property Type is String, this value will be passed to the Tool.

gameObjectInput *GO*

If Property Type is Game Object, this value will be passed to the Tool. Note you are allowed to pass an empty Game Object reference value.

scriptInput *MonoBehaviour*

If Property Type is Script, this value will be passed to the Tool.

findInputByName *bool*

If true, and Property Type is either Game Object or Script, the proper input field will be populated by the game object with the unique name indicated by Input Name.

inputName *string*

The unique name of the Game Object that is to be used as Game Object Input, or that contains the script to be used as Script Input.

resetOnComplete *bool*

If true, this game object will be deactivated and ready for re-activation. If false, this tool will be disabled instead.

HUD Manager

Tool: at_HUDManager

Description:

Manages a number of HUD elements. Options for image, viewport position, scale and transitions of the images.

Purpose:

To display general HUD images, to indicate game progression changes.

Properties:

hudTextures *Texture[]*

A list of textures to display on screen upon trigger.

hudDepth *int*

The layer order of this HUD Manager. The larger the number, the more behind the layer is configured to display textures.

hudTop *float*

The top edge of the texture, expressed as a ratio of vertical screen space, which is typically a number between zero and one.

hudLeft *float*

The left edge of the texture, expressed as a ratio of horizontal screen space, which is typically a number between zero and one.

hudWidth *float*

The width of the texture, expressed as a ratio of horizontal screen space, which is typically a number between zero and one. The height of the texture will remain in proportion.

fadeDuration *float*

The duration of the fade in seconds.

fadeColor *Color*

The color representing the target to fade to. (Default: Color.clear)

fadeTop *float*

The target top to fade to.

fadeTopCurve *AnimationCurve*

Optional animation curve for the top edge to follow during the fade. If none is defined, will interpolate linearly.

fadeLeft *float*

The target left to fade to.

fadeLeftCurve *AnimationCurve*

Optional animation curve for the left edge to follow during the fade. If none is defined, will interpolate linearly.

fadeWidth *float*

The target width to fade to.

fadeWidthCurve *AnimationCurve*

Optional animation curve for the width to follow during the fade. If none is defined, will interpolate linearly.

HUD Trigger

Tool: at_HUDTrigger

Description:

Will call a HUD Manager game object in the scene to switch the current HUD item.

Purpose:

To cause HUD elements to change either by collision trigger or timed event.

Properties:

hudManager *GO*

The HUD Manager object to call when this HUD Item is triggered.

hudItem *int*

The index of the HUD Manager's HUD Texture element to display. Like all arrays, the first element has an index of zero.

hudFade *bool*

If true, perform the fade on this HUD Item as specified on the HUD Manager. If false, display of the HUD Item instantly.

reverseFade *boo*l

If true, perform the fade in reverse.

resetOnTrigger *bool*

If true, will deactivate this game object upon trigger and repeat upon re-activation. If false, will disable this component instead.

Log Trigger

Tool: at_LogTrigger

Description:

Will send configured strings to the game log. Options to timestamp, log on deactivation of this game object, continuous logging, and logging only changes in this game objects' activation state.

Purpose:

To track things that happen in your trigger system and print it out to the game log for review.

Properties:

logString *string*

Text to print to the editor console and log file when this tool is activated.

logOnDeactivate *string*

Text to print when this tool is deactivated.

timestamp *bool*

If true, print the time in seconds since level load as a prefix to the Log String.

continuousLogging *bool*

If true, check for conditions to print the Log String each frame.

logChangeOnly *bool*

If true, only print the Log String when the activation state of this tool changes. This can be useful if Continuous Logging is true.

Menu Manager

Tool: at_MenuManager

Description:

Displays interactive buttons and text, with functionality to hide/show buttons based on button pressed state and to input text. Multiple options for style, position size and behavior of buttons. Options for text display and input, as well as a separate text display on button hover.

Purpose:

A game menu system that receives and holds data from user input, such as game options, and works with a Scene Manager to facilitate basic menu item handling.

Properties:

MenuButton *class*

buttonLabel *string*

The name of the button and text of the button label.

buttonAltLabel *string*

The toggled button label displayed when Button Pressed is true.

buttonTooltip *string*

The text to display in the tooltip area when the cursor hovers over this button.

buttonTop *float*

The top edge of the button, expressed as a ratio of vertical screen space. Typically, a number between zero and one.

buttonLeft *float*

The left edge of the button, expressed as a ratio of horizontal screen space. Typically, a number between zero and one.

buttonHeight *float*

The height of the button, expressed as a ratio of vertical screen space. Typically, a number between zero and one.

buttonWidth *float*

The width of the button, expressed as a ratio of horizontal screen space. Typically, a number between zero and one.

buttonPressed *bool*

The current toggle state of this button control. This is automatically updated at runtime.

conditionalDisplay *bool*

If true, this button will display only if Display Button is toggled on.

displayButton *int*

If Conditional Display is true, the button control at this index must be toggled on for this button to display.

conditionalEnabled *bool*

If true, this button will display with the Disabled Button Style and be disabled if any of the Disabled Buttons are toggled on.

disableButtons *int[]*

A list of button indexes that can disable this button if pressed.

buttons *MenuButton[]*

Each button control on the menu.

buttonFontSize *int*

The font size the button labels should appear as at a screen width of 1024 pixels. This value will override the font size defined in its GUIStyle.

buttonStyle *GUIStyle*

The GUI Style defining the button appearance when enabled.

disabledButtonStyle *GUIStyle*

The GUI Style defining the button appearance when disabled.

buttonPressSfx *AudioClip*

The Audio Clip to play upon button press if the signal was received by the Scene Manager. This audio can be configured to mute per the Scene Manager's SfxOption toggle control.

MenuText *class*

defaultText *string*

Text to display in this control by default.

modifiedText *string*

The current text. This is automatically updated at runtime.

readOnly *bool*

If true, this text control is for display purposes only. If false, the text is editable.

textTop *float*

The top edge of the text control, expressed as a ratio of vertical screen space.

textLeft *float*

The left edge of the text control, expressed as a ratio of horizontal screen space.

textHeight *float*

The height of the text control, expressed as a ratio of vertical screen space.

textWidth *float*

The width of the text control, expressed as a ratio of horizontal screen space.

text *MenuText[]*

Each text control and display on the menu.

textFontSize *int*

The font size the read-only text should appear as at a screen width of 1024 pixels. This value will override the font size defined in its GUIStyle.

textReadOnlyStyle *GUIStyle*

The GUIStyle defining the appearance of read-only text.

textInputFontSize *int*

The font size the input text should appear as at a screen width of 1024 pixels. This value will override the font size defined in its GUIStyle.

textInputStyle *GUIStyle*

The GUIStyle defining the appearance of text input.

tooltipTop *float*

The top edge of the tool tip display, expressed as a ratio of vertical screen space.

tooltipLeft *float*

The left edge of the tool tip display, expressed as a ratio of horizontal screen space.

tooltipHeight *float*

The height of the tool tip display, expressed as a ratio of vertical screen space.

tooltipWidth *float*

The width of the tool tip display, expressed as a ratio of horizontal screen space.

tooltipFontSize *int*

The font size the tooltips should appear as at a screen width of 1024 pixels. This value will override the font size defined in its GUIStyle.

tooltipStyle *GUIStyle*

The GUIStyle defining the appearance of tooltips.

Menu Trigger

Tool: at_MenuTrigger

Description:

Will perform a standard control action when a menu button is pressed.

Purpose:

To serve as a controller for auxiliary menu tools not configured with a Scene Manager.

Properties:

ButtonAct *enum*

ActivateOnPress	The target object will be activated when the button is pressed.
ActivateOnUnPress	The target object will be activated when the button is un-pressed.
ActiveMatchPress	The target object will be set active to match the button pressed state.
ActiveOppositePress	The target object will be set active opposite to the button pressed state.

ButtonControl *class*

buttonLabel *string*

The button label as found on the Menu Manager.

buttonAction *ButtonAct*

The button action to monitor. This also governs how to act upon the Target Object, based on the pressed state of the button.

activateObject *GO*

The object to act upon once the button action is detected. The action will either activate or deactivate the object, based on the Button Action and the pressed state of the button.

menuManager *at_MenuManager*

The Menu Manager object that defines the buttons this tool will monitor.

buttonControls *ButtonControl[]*

A list of controls matching the list of buttons found on the Menu Manager.

Type Box

Tool: at_TypeBox

Description:

Will display a popup-style dialog box with text that can type letter-by-letter. Multiple options available for text style, typing rate and variance, user skipping, and typing sound effects. Options available for box depth, background image and to allow tool's game object transforms to be inherited for animation of the box position and scale.

Purpose:

To display introduction text, story set-up or in-game dialog from characters.

Properties:

textToDisplay *string*

The text to display. Rich Text markup will apply. (<br> for new line, <b>...</b> to bold, etc.)

displayDelay *float*

The length in seconds from the time this tool is activated until the typing begins. (Default: 0.5)

letterTypeInterval *float*

The time in seconds between typing letters of the text. (Default: 0.1)

typeIntervalVariance *float*

The amount of time in seconds that Letter Type Interval can vary. (Default: 0.075)

allowSkip *bool*

If true, will allow the user to skip the text typing with any input from mouse or keyboard. First, with accelerated typing, then with instant display.

textBox *Rect*

The display rectangle the text box will appear. This is measured in proportions of screen space; typically zero to one. (Default: X 0.3, Y 0.2, W 0.4, H 0.4)

inheritTransforms *bool*

If true, the Text Box will inherit this object's transform values, including position x and y, z for depth, and scale x and y for width and height.

background	*Texture2D*

An optional background texture to use. If none is specified, the default GUI box texture will be used.

textFont	*Font*

An optional text font to display. If none is specified, the default Arial font will be used.

fontSize	*int*

The font size of the text as displayed at a screen width of 1024 pixels. The font will scale appropriately. (Default: 20)

textColor	*Color*

The default text color. Note that this can be altered within the text using Rich Text markup. (Default: Color.white)

textPadding	*float*

The amount of padding around the text. This is measure in proportions of screen space; typically zero to 0.5. (Default: 0.025)

letterTypeSFX	*AudioClip*

An optional audio clip to play when each letter is typed. (Will omit spaces)

activateOnComplete	*GO*

An optional game object to activate when the text typing is complete.

Loading Screen Switch

Tool: at_LoadingScreenSwitch

Description:

Switches the texture used on the Scene Manager during scene load.

Purpose:

To change the loading screen image for the next scene switch.

Properties:

loadingScreenTexture *Texture*

The texture to set as the Scene Manager's loading screen.

Lock Game Object

Tool: at_LockGameObject

Description:

An editor tool to force a game object to remain at world origin with a default rotation and scale.

Purpose:

To lock a game object in place, in editor and at runtime, so that it may be safely used as a folder, and be a parent to other game objects for effective and reliable scene organization.

Properties:

This tool contains no editable properties.

Options Manager

Tool: at_OptionsManager

Description:

Destroys objects based on options set through use of a Menu Manager.

Purpose:

To modify a scene’s trigger system at runtime based on options set by the player.

Properties:

OptionConfig *class*

optionLabel *string*

The matching label to a button control on the Menu Manager that dictates the enabled state of this option. If the button control is toggled on, this option is considered enabled.

optionEnabled *bool*

The current enabled state of this option. This is automatically reconfigured at runtime.

optionOnDestroy *GO[]*

A list of objects to destroy if this option is enabled.

optionOffDestroy *GO[]*

A list of objects to destroy if this option is disabled.

options *OptionConfig[]*

The options available in this scene, as defined by trigger system objects that are destroyed based on a toggle control defined in the Menu Manager.

Scene Attraction

Tool: at_SceneAttraction

Description:

A delayed scene trigger that disables any active menu manager in the scene prior to switching.

Purpose:

To act as a simple ‘Attract Mode’ manager, particularly when game is unused and scene switching from main menu to sub menus, demo play, or the like is desirable.

Properties:

attractScene *string*

The name of the scene to load after the Attract Delay. Note that this scene must be included in the project Build Settings.

attractDelay *float*

The time in seconds after activation to load the Attract Scene.

Scene Manager

Tool: at_SceneManager

Description:

Will handle scene switching as called by a Scene Trigger, and will coordinate Music Manager, Captions Manager and Menu Manager objects over multiple scenes within a game. Options include a loading screen image.

Purpose:

To manage multiple scenes in your game, and allow options and menu settings to be consistent.

Properties:

ButtonAction *enum*

Toggle	The button is intended to toggle between two states.
Cycle	The button is intended to cycle around a list of button states, found in Label Names.
LoadLevel	The button is intended to immediately load a scene when pressed.
QuitGame	The button is intended to quit the application, when built.

OptionMap *enum*

NoOption	This button handles no option.
SFX	This button toggles the sound effects, via muting all objects with SFX Object unless Voice, Interface or Movie type SFX Objects are used.
Music	This button toggles the music, via muting the Music Object and any Music type SFX Objects.
Voice	This button toggles the voice audio, via muting all Voice type SFX Objects.
Interface	This button toggles the interface audio, via muting all Interface type SFX Objects
Movie	This button toggles the movie audio, via muting all Movie type SFX Objects.
Captions	The button toggles display of the captions by controlling Captions Object.
Custom	This button toggles the state of a custom option, handled in individually loaded scenes via the Options Manager, which looks for this control by name.

ButtonHandle *class*

buttonLabel *string*

The name of the button control. This label must match that of the button control on the Menu Manager.

buttonPressed *bool*

The current toggle state of the button control, as automatically relayed by the Menu Manager.

buttonAction *ButtonAction*

The button action type associated with this control.

labelNames *string[]*

A list of button label names that will be cycled through if Button Action is Cycle. Option Managers in scene can expect to only find the selected label, and that Option Enabled value will always be true.

levelName *string*

The name of the scene to load, if Button Action is Load Level. This scene must be included in the Build Settings scene list to be available for loading.

option *OptionMap*

The option type this control represents. Menu sound effect, music, voice, interface, movie audio and captions options are available by default. Custom options can be implemented as button controls that the Options Manager refers to by label name.

menu *at_MenuManager*

The Menu Manager object that defines the button controls and input text to be tracked between scenes. If this is undefined, it will look for any Menu Manager object in the scene at runtime.

controls *ButtonHandle[]*

Each button control that is tracked between scenes, as defined by the Menu Manager and implemented by the Options Manager. Although controls can be defined here, the full list of button controls will be populated automatically at runtime.

textState *string[]*

A list of text labels matching the modified text of each text control on the Menu Manager. This is automatically updated at runtime.

musicObject *GO*

The Music Manager object used to manage music. If none is defined, one will be automatically created at runtime.

captionsObject *GO*

The Captions Manager object used to play captions. If none is defined, one will be automatically created at runtime.

loadingScreen *Texture*

An optional texture to display over the viewport during level loading.

Scene Trigger

Tool: at_SceneTrigger

Description:

Will call a Scene Manager for the purpose of switching scenes.

Purpose:

To switch to the next scene in your game.

Properties:

sceneName *string*

The name of the scene to load. This scene must be included in the Build Settings scene list to be available for loading.

relayDelay *float*

Time in seconds from activation of this tool until the scene is loaded.

Screenshot Manager

Tool: at_ScreenshotManager

Description:

A material switcher to override prefab/mesh texture configurations at runtime. Configurable mesh, material and material slot reference.

Purpose:

To specify arbitrary material configuration on prefab characters or miscellaneous game objects.

Properties:

StorageLocation *enum*

Desktop	Screenshot image files will save to the Desktop for this account.
MyDocuments	Screenshot image files will save to the MyDocuments folder for this account.
MyComputer	Screenshot image files will save to the MyComputer folder.
MyPictures	Screenshot image files will save to the MyPictures folder for this account.
Personal	Screenshot image files will save to the Personal folder.

screenshotKey *string*

The key code name of the input key to capture a screenshot. This uses the KeyCode format found in Unity script documentation. (Default: "f12")

screenshotBaseName *string*

Base name for screenshot PNG image files. The full name will include a suffix with a screenshot index number. (Default: "Game")

includeTimeStamp *bool*

If true, the file name will include a timestamp to indicate month, day, hour and minute. If false, only a screenshot index number will be included. (Default: true)

screenshotSaveTo *StorageLocation*

Location to store screenshot images on Windows machines. Mac machines will store to named account folder using MyDocuments.

Singleton By Name

Tool: at_SingletonByName

Description:

Identifies this game object as one that should not be deleted upon scene switching, and as one that should not allow a duplicate to exist in the same scene, based on this object's name.

Purpose:

To keep those objects that should remain loaded from scene to scene, throughout the game. Note: This will keep the game object and all its children from being deleted. It is recommended that tools meant to travel between scenes be parented under a game object with this tool.

Properties:

This tool contains no editable properties.

Tool Index