Amazon Sumerian

User Guide

Amazon Sumerian: User Guide

Table of Contents

What Is Amazon Sumerian?

Amazon Sumerian is a set of tools for creating high-quality virtual reality (VR) experiences on the web. With Sumerian, you can construct an interactive 3D scene without any programming experience, test it in the browser, and publish it as a website that is immediately available to users.

> **Limited Preview**
> Amazon Sumerian is in limited preview. Sign up to apply for access: Amazon Sumerian Preview

Use the Sumerian library of assets or bring your own. When you import 3D models, Sumerian converts and optimizes them automatically. Sumerian also has a library of primitive shapes, 3D models, hosts, textures, and scripts.

> **Note**
> New to 3D, VR, animation, and scripting? The Sumerian website has a ton of helpful tutorials for every level of experience.

The Sumerian 3D engine provides a library for advanced scripting with JavaScript, but you don't have to be a programmer to create interactive VR! Use the built-in state machine to animate objects and respond to user actions like clicks and movement.

When you're ready to share your work with the world, you can publish it directly to Amazon CloudFront as a static website that can be viewed with any WebVR-compatible browser and headset.

Amazon Sumerian Use Cases and Requirements

At the core of Amazon Sumerian is a web-based editor for constructing 3D scenes with animation, scripted interaction, and special effects. The editor runs in your web browser, and all of your data is stored in AWS. The editor outputs scenes to Amazon CloudFront as a static website that you can load directly into any WebVR-compatible browser and headset, or embed in your website for others to access.

> **Note**
> Don't know how to script? The Sumerian editor provides a fully featured state machine for scripting animations and user interactions visually, with no coding required.

WebVR is an open specification that lets you create and share virtual reality (VR) experiences through the web. WebVR applications, like any web app, are supported on several desktop and mobile operating systems. This enables you to avoid the need to port your application to different programming languages and package formats to reach all users. You can use WebVR on your phone with a fold-together cardboard headset, or to a limited degree without any headset at all. See the WebVR website for a list of compatible headsets and browsers: webvr.info.

Sumerian also lets you create augmented reality (AR) applications. An AR application can use your phone's camera or an AR-compatible headset to overlay graphics on the real world. Sumerian provides a template for creating ARKit applications for iOS phones. For other devices, you can script against the Sumerian engine JavaScript library to perform the required object tracking and motion controls.

Sumerian provides a library of optimized 3D objects and scene templates that you can use to construct scenes without any existing assets. If you do have 3D models, you can import them with their animations and textures by dragging them from your file system into the editor canvas. Sumerian supports models in `OBJ` and `FBX` formats.

Amazon Sumerian Concepts

Amazon Sumerian lets you create virtual reality (VR) and augmented reality (AR) *scenes* that are made up of *components* and *entities*, organized into *projects*. Let's look closely at the concepts used in the Sumerian editor and this guide.

Scenes

A scene is a 3D space that contains objects and behaviors that define a VR or AR environment. Objects include geometry, materials, and sounds that you import from a supported file format, and objects that you create in the scene like lights, cameras, and particle effects. Behaviors include state machine behaviors, animations, timelines, and scripts.

When you're ready to show off your scene, export it directly to Amazon CloudFront as a static website that you can open in a browser.

See Scenes (p. 8) for more information.

Components and Entities

All objects and behaviors are *components* that combine to create *entities*. For example, when you import a 3D model and add it to a scene, the editor creates an entity that has a geometry component, a material component, a transform component, and an animation component. You can then use the editor to add a rigid body, colliders, and other components to the entity.

See Entities (p. 32) for more information.

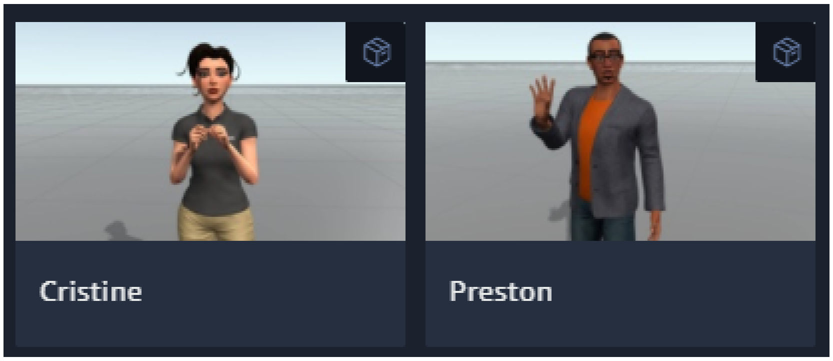
Assets

Assets are the images, sounds, scripts, models, and documents that you import into Sumerian to use in a scene. You can manage assets independently of the scenes that use them in the *asset library*. Assets can belong to a user or project.

See Asset Packs (p. 9) for more information.

Hosts

A host is a asset provided by Sumerian that has built in animation, speech, and behavior for interacting with users.



Hosts use Amazon Polly to speak to users from a text source. You can use hosts to engage users and guide them through a VR experience.

See The Amazon Sumerian Host Component (p. 36) for more information.

Projects

Projects are an organizational tool for managing scenes, assets, and templates.

See Projects (p. 7) for more information.

Templates

Templates let you save a copy of a scene to use as a starting point for other scenes. Templates belong to a project. Sumerian provides several templates, which you can access from the dashboard.

See Templates (p. 10) for more information.

Amazon Sumerian Permissions

You can use AWS Identity and Access Management (IAM) to grant Sumerian permissions to users and compute resources in your account. IAM controls access to AWS at the API level to enforce permissions uniformly and securely.

To use the Sumerian editor (p. 12), add the following policy to your IAM user.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sumerian:*"
            ],
            "Resource": "*"
        }
    ]
}
```

You only need access to Sumerians APIs. Sumerian manages all of the storage (Amazon S3) and content delivery (Amazon CloudFront) related to the scenes that you create outside of your account.

To use AWS services in a scene, the scene needs credentials as well. You can use Amazon Cognito Identity to create an identity pool that gives the scene access to a role with permission to use AWS. Create a role that has permissions to any services that you will access from scripts, and permissions for components that use AWS services.

To create an identity pool for a Sumerian scene

1. Open the **Federated identities** page in the Amazon Cognito console.
2. Choose **Create new identity pool**.
3. Create a pool with the following settings.
   - **Unauthenticated identities** – enabled
4. Choose **Edit identity pool** to see the pool details.
5. Note the **Identity pool ID** for later use.

When you create an identity pool, Amazon Cognito prompts you to create two roles, an authenticated role, and an unauthenticated role. Add permissions to the unauthenticated role.

To add permissions to an identity pool role for a Sumerian scene

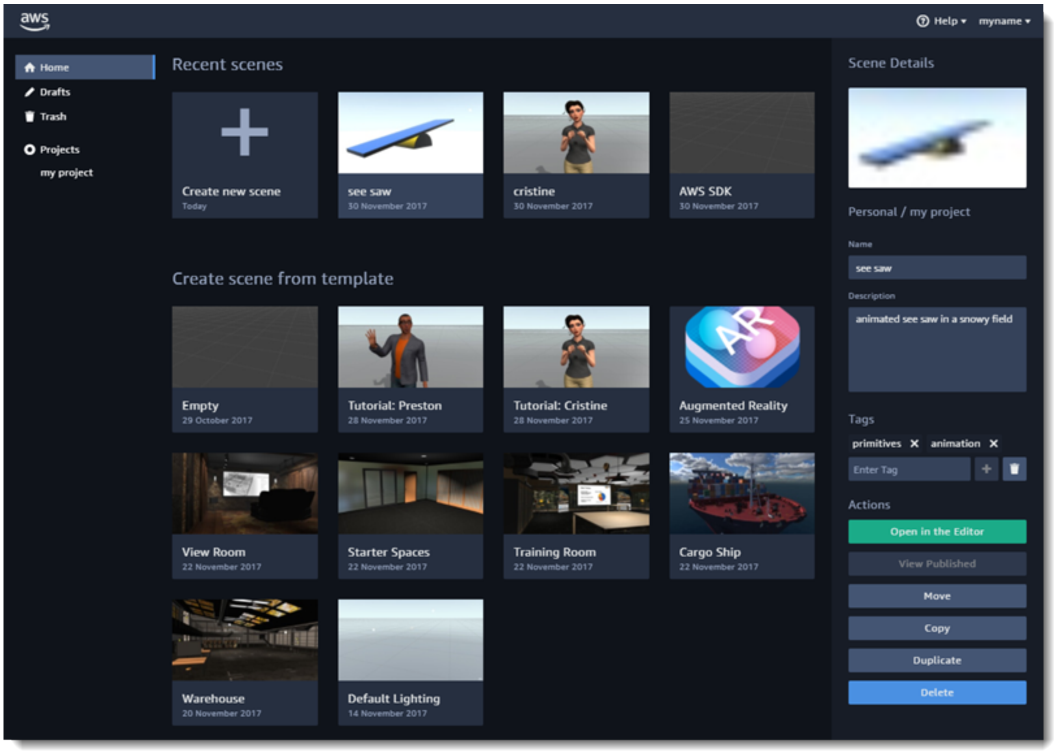
1. Open the **Roles** page in the IAM console.
2. Choose the role named **Cognito_*pool-name*Unauth_Role**.
3. Choose **Attach policy** and add policies for the services that your scene uses.
   - **Speech** – `AmazonPollyReadOnlyAccess` gives the scene permission to use Amazon Polly to render text into audio with the speech component (p. 37).

     AWS SDK for JavaScript – add policies that grant access to the services that you call with the SDK for JavaScript.

Assign the identity pool to your scene under **AWS configuration (p. 22)** in scene settings.

The Amazon Sumerian Dashboard

The Dashboard is the first thing you see when you open the Amazon Sumerian app. This is where you manage your projects, scenes, asset packs, and templates.



Personal / my project

Name: see saw

Description: animated see saw in a snowy field

Tags: primitives ✕ animation ✕

Projects collect scenes and the templates and asset packs that you export from them. You can create draft projects outside of a project, but you must have a project to export templates and assets.

When you open a scene in the editor, it is locked to prevent other users from modifying it. The dashboard manages locks and lets you steal a lock if the other user leaves a scene open by accident.

Topics

Projects

Projects collect the scenes that you are working on.

To create a project

1. Open the Sumerian dashboard.
2. Choose **Projects**.
3. Choose **New project**.
4. Enter a project name and choose **Create**.

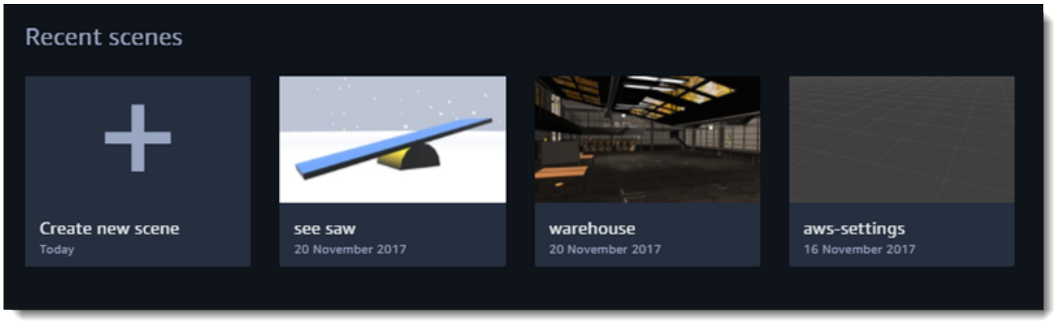
Once you have a project, you can use the dashboard to make a copy or delete it.

To manage a project

1. Open the Sumerian dashboard.
2. Choose a project.
3. Under **Project details**, use one of the following options.
   - **Thumbnail** – Choose **Browse** to upload a thumbnail image.
   - **Name** – Change the project name.
   - **Description** – Change the project description.
   - **Actions** – **Move** or **Copy** the project. **Delete** the project to send it to the **Trash**.
   - **Published URLs** – Choose **View URL List** to get links to all of the project's scenes that have been published in Amazon CloudFront.

Scenes

A scene is a 3D space that you manage in the dashboard and work on in the Sumerian editor. Sumerian provides several templates that you can use as a starting point.



Scenes can be drafts, or part of a project.

To create a scene

1. Open the Sumerian dashboard.
2. Choose the location to create the scene.
   - **Home** – Create a draft scene.
   - **Drafts** – Create a draft scene.
   - **Project** – Create a scene in one of your projects.
3. Choose **Create scene**.
4. (optional) Choose a template (p. 10).
5. Enter a scene name and choose **Create**.

When you create a scene, it opens in the Sumerian editor (p. 12) for immediate use. Once you have a scene, you can use the dashboard to make a copy or delete it. Choose the Sumerian icon in the upper left corner to leave the scene and return to the dashboard.

To manage a scene

1. Open the Sumerian dashboard.
2. Locate your scene under **Recent scenes**, **Drafts**, or a project.
3. Choose the scene by clicking its thumbnail.

   Note
   If you click on the name of the scene or double-click the thumbnail, the scene opens in the Sumerian editor.
4. Under **Scene details**, use one of the following options.

   - **Thumbnail** – Choose **Browse** to upload a thumbnail image.
   - **Name** – Change the scene name.
   - **Description** – Change the scene description.
   - **Tags** – Add tags to the scene for use with filters.
   - **Actions**
     - **Open** – Open the scene in the Sumerian editor.
     - **View published** – Open the published version of the scene hosted in Amazon CloudFront.
     - **Move** – Move the scene to a different project.
     - **Copy** – Copy the scene to a different project.
     - **Duplicate** – Create a copy of the scene in the same project.
     - **Delete** – Send the scene to the **Trash**

Additional options for scenes are available in the Sumerian editor scene settings (p. 20).

Asset Packs

The **Assets** page for a project shows asset packs that have been exported from a scene.

In the dashboard, you can change the name and description of a pack, and copy or move it to another project.

To manage an asset pack

1. Open the Sumerian dashboard.
2. Choose a project.
3. Choose **Assets**.
4. Choose an asset pack.
5. Under **Asset details**, use one of the following options.

   - **Thumbnail** – Choose **Browse** to upload a thumbnail image.
   - **Name** – Change the asset pack name.
   - **Description** – Change the asset pack description.
   - **Tags** – Add tags to the asset pack for use with filters.
   - **Actions**
     - **Move** – Move the asset pack to a different project.

  - **Copy** – Copy the asset pack to a different project.
  - **Delete** – Send the asset pack to the **Trash**

Additional options for asset packs are available in the Sumerian editor (p. 53).

Templates

Templates are scenes that have been exported from a project for use as a starting point for other scenes. In addition to the templates provided by Sumerian, the dashboard lets you manage templates that you have exported from a scene.

You can use the dashboard to create a scene from a template, or move or copy templates between scenes. Sumerian also provides a library of templates.

To create a scene from a template

1. Open the Sumerian dashboard.
2. Choose **Create new scene**.
3. Choose one of the **Sumerian Templates**, or choose **My templates** to use a template from one of your projects.
4. Enter a name for your scene and choose **Create**.

Create templates from your scenes from the scene settings section (p. 20) in the Sumerian editor. You can then copy your templates to other projects from the **Templates** section of the scene's project page in the dashboard.

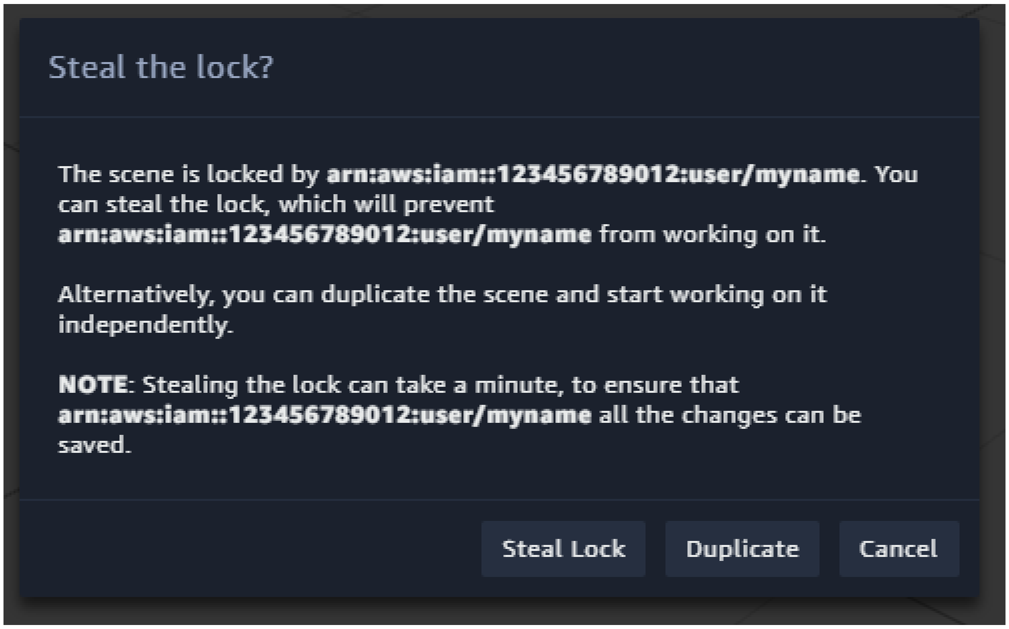
To manage a template

1. Open the Sumerian dashboard.
2. Choose a project.
3. Choose **Templates**.
4. Choose a template.
5. Under **Template details**, use one of the following options.

   - **Thumbnail** – Choose **Browse** to upload a thumbnail image.
   - **Name** – Change the template name.
   - **Description** – Change the template description.
   - **Tags** – Add tags to the template for use with filters.
   - **Actions**
     - **Move** – Move the template to a different project.
     - **Copy** – Copy the template to a different project.
     - **Delete** – Send the template to the **Trash**

Locks

The Amazon Sumerian editor uses locks to control modifications to a scene. When you open a scene, the editor creates a lock on the scene and refreshes it periodically. If you try to open the scene in a different browser while the lock is active, you will see an error.



Steal the lock?

The scene is locked by **arn:aws:iam::123456789012:user/myname**. You can steal the lock, which will prevent **arn:aws:iam::123456789012:user/myname** from working on it.

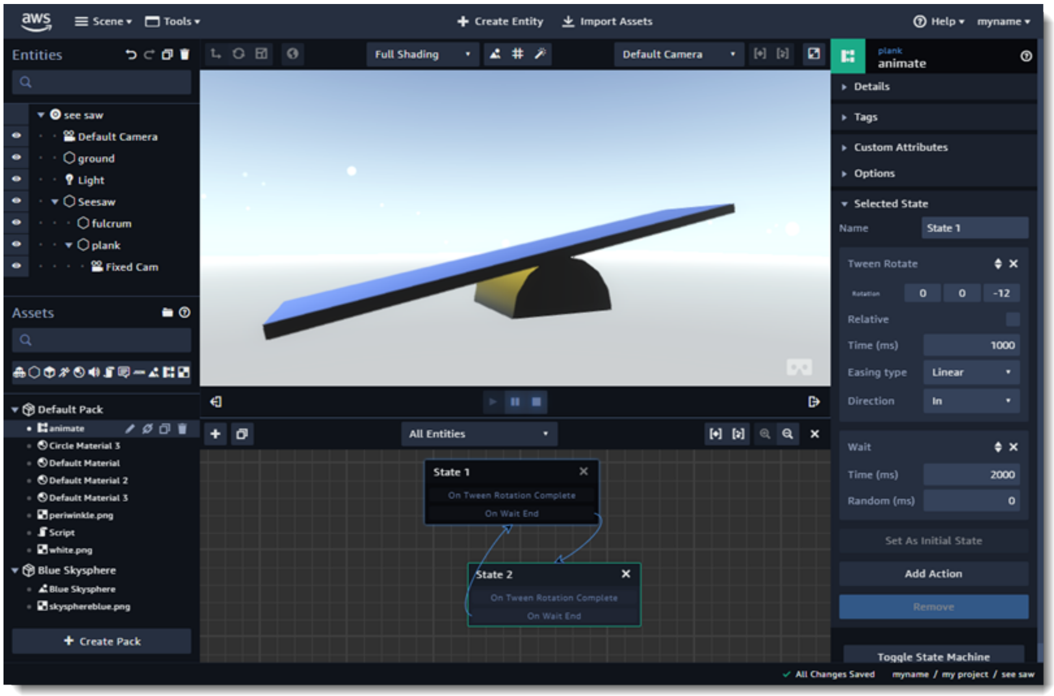
Alternatively, you can duplicate the scene and start working on it independently.

NOTE: Stealing the lock can take a minute, to ensure that **arn:aws:iam::123456789012:user/myname** all the changes can be saved.

You can force Sumerian to discard the lock if you are sure that no one else is working on the scene, or create a copy of the scene and work on that.

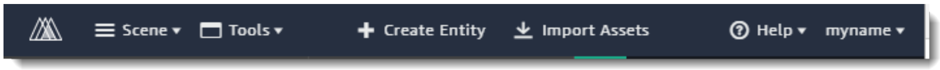
Editor

The Sumerian editor provides an interface for easily importing assets, building a scene, and publishing it to Amazon CloudFront.



When you load a scene in the Sumerian editor, you can see a menu bar at the top of the screen, the entities panel, the assets panel, the canvas, and the inspector panel. The menu bar at the top of the screen provides menus for navigating between scenes, editing tools, and publishing.

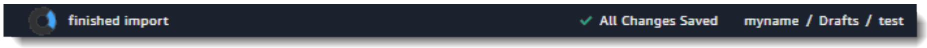
Top bar



- *Logo* – Exit to dashboard.
- **Scene** – create a new scene, publish your scene, or open a recent scene.
- **Tools** – access the text editor, behavior editor, and timeline editor.
- **Create entity** – add a shape, light, camera, or blank entity to the scene.
- **Import assets** – open the asset library.
- **Help** – view the shortcut list or submit feedback.
- *Username* – log out.

The status bar at the bottom of the screen shows updates about save, import, and rendering operations.

Status bar


finished import

✓ All Changes Saved   myname / Drafts / test

- *Progress bar* – Shows information about the current activity, such as model uploading.

- *Path* – The current user, project and scene.

The following topics describe the menu options in each of the areas of the editor.

Panels and Menus

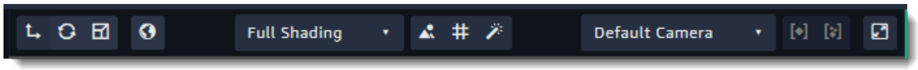
- The Amazon Sumerian Editor Canvas (p. 13)
- Using the Asset Library in the Amazon Sumerian Editor (p. 14)
- Using the Assets Panel in the Amazon Sumerian Editor (p. 15)
- Using The Entities Panel in the Amazon Sumerian Editor (p. 16)
- Using the Inspector Panel in the Amazon Sumerian Editor (p. 16)
- Keyboard and Mouse Controls for the Amazon Sumerian Editor (p. 17)
- Publishing Scenes in the Amazon Sumerian Editor (p. 19)

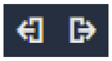
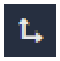
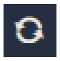
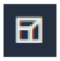
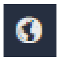
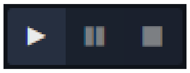
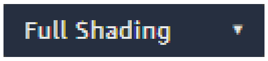
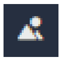
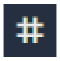
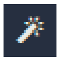
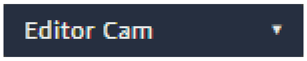
The Amazon Sumerian Editor Canvas

In the center of the editor, the WebGL-rendered viewport is located. Here you can navigate, inspect and preview the contents of your scene.

The menu bar at the top of the canvas has options for camera, playback, and rendering. Many of the buttons also have equivalent keyboard commands (p. 17).

Canvas Menu



-  – hide or show side panels.
-  – change the transform handles to translate mode.
-  – change the transform handles to rotation mode.
-  – change the transform handles to scale mode.
-  – switch between relative and absolute positioning.
-  – preview the rendered scene in the canvas.
-  – choose the render mode for the canvas.
-  – show or hide the skybox texture.
-  – show or hide the grid.
-  – show or hide post effects.
-  – view the scene with a preset camera.

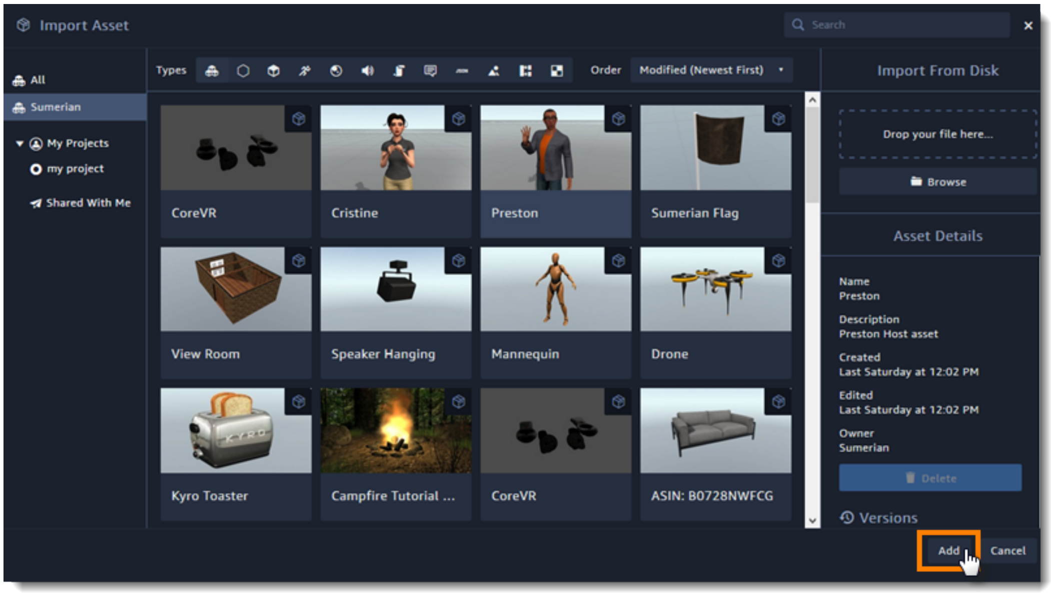
- – fill the canvas with the selected entity.
- – fill the canvas with all entities in the scene.
- – fill the screen with the canvas.

Using the Asset Library in the Amazon Sumerian Editor

You can use the Amazon Sumerian Editor's asset library to import assets from the Sumerian library, your local machine, or from asset packs (p. 53) that you have exported from a scene.

To import assets

1. Open a scene in the Sumerian editor.
2. Choose **Import assets**.
3. Choose an asset type (p. 52) to filter the available assets by type.
4. Choose an asset pack and then choose **Add** to add it to your scene's assets.



Name
Preston

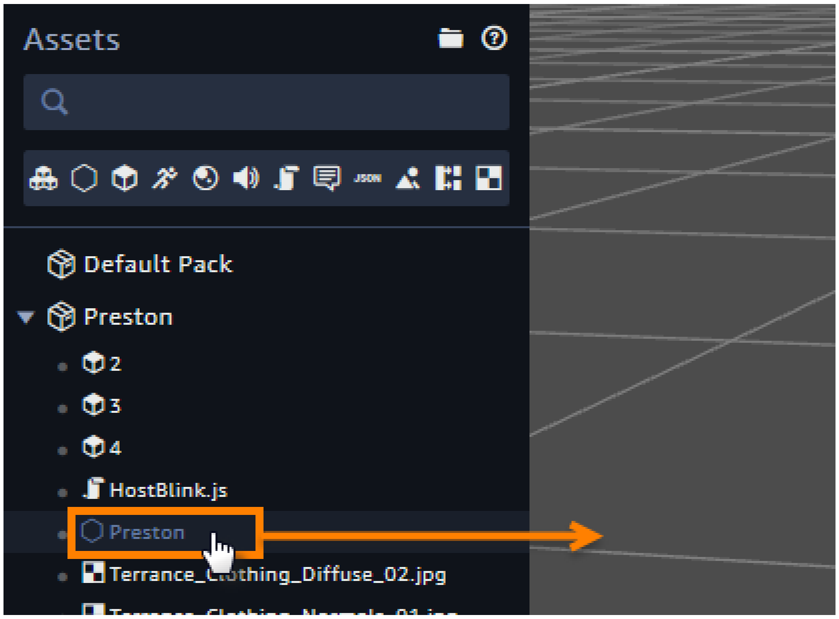
Description
Preston Host asset

Created
Last Saturday at 12:02 PM

Edited
Last Saturday at 12:02 PM

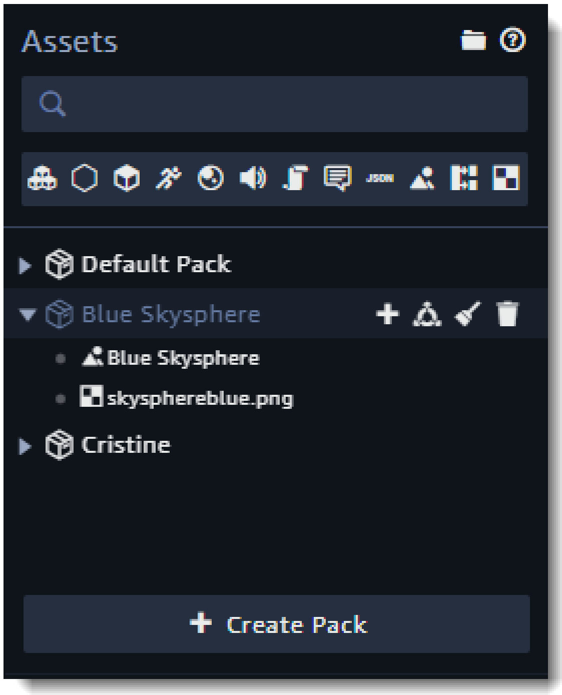
Owner
Sumerian

5. After the editor finishes importing the asset pack, drag an entity that it contains from the assets panel (p. 15) onto the canvas to add it to your scene.



Using the Assets Panel in the Amazon Sumerian Editor

The assets panel shows all assets belonging to the scene. Assets are portable versions of entities or entity components, and can be created from external files or from entities that you create within the editor.

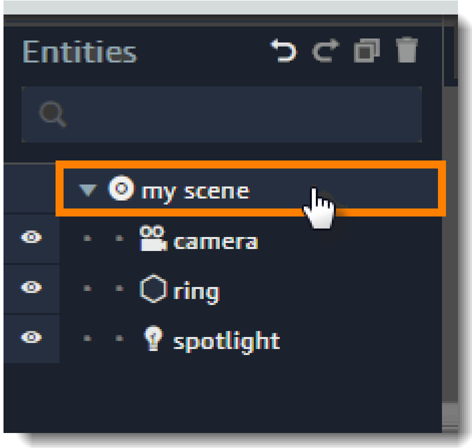


To create an asset, drop a file from your computer, or an entity from the entities panel, onto the assets panel. Files may be split into multiple assets depending on the type of file.

For more information, see Assets (p. 52).

Using The Entities Panel in the Amazon Sumerian Editor

The entities panel shows you scene's entities in a hierarchy, starting with the scene itself. An entity can be a child of the scene or of another entity. When you choose the scene in the entities panel, the inspector panel (p. 16) shows scene settings. When you choose an entity, the inspector panel shows the entity's components.



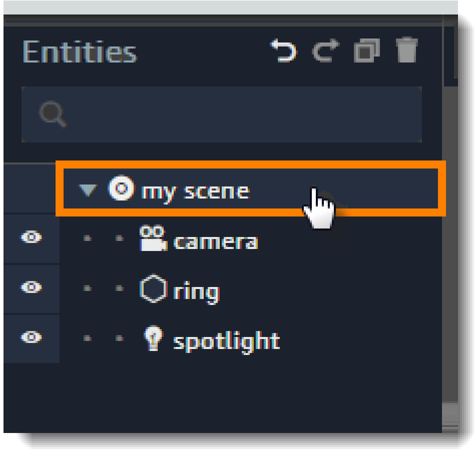
Organize your entities by their physical or logical relationship to other entities. An entity's position, rotation, and scale are relative to its parent. When you move the parent, the child moves as well. To change an entity's parent, drag it onto the new parent in the entities panel.

Entities panel controls

- ▼ – collapse the entity to hide its children in the entities panel.
- 👁 – hide or show an entity in the canvas.
- ⧉ – duplicate an entity.
- 🗑 – delete an entity.
- ↶ ↷ – undo or redo changes.

Using the Inspector Panel in the Amazon Sumerian Editor

Use the inspector panel to manage scene settings, entities, and assets. When you select any of these elements in the Sumerian editor, you get the following properties in a section named after the element.



Generic properties

- **Thumbnail** – the thumbnail image for the element. Drop an image onto the thumbnail field or hover your mouse over it and choose **Take screenshot** to save an image of the current view of the canvas.
- **Name** – the name of the element.
- **ID** (read-only) – a unique identifier for the element.
- **Type** (read-only) – the type of element. `scene`, `entity`, or an asset type (p. 52).
- **Description** – description of the element.
- **Tags** – key-only metadata that you can use in scripting. You can read tags or search for entities with specific tags by using the context object (p. 60).
- **Custom attributes** – key-value metadata that you can use in scripting. You can read attributes by using the context object (p. 60).

When you choose the scene in the entities panel (p. 32), or click on the background of the scene in the canvas, the inspector panel shows several additional sections for settings that apply to the entire scene, like environmental settings, post processing effects, and AWS SDK credentials. See ??? (p. 20) for more information.

When you choose an entity in the entities panel, or click on it in the canvas, the inspector panel shows a section for each component on the entity. At a minimum, every entity has a transform component that determines its location, rotation, and size. Entities created by dropping assets onto the scene have additional components based on their type, and you can add components to any entity in the inspector panel by choosing **Add component** at the bottom of the panel. See ??? (p. 32) for more information.

When you choose an asset in the assets panel (p. 15), the inspector panel shows sections for only components that apply to every instance of the asset in the scene. For example, a script asset only has code, but a script component on an entity can have parameters that customize that instance of the script. A material asset, on the other hand, has all of the material component (p. 34) properties. Modifying any of these properties changes every instance of the material in the scene.

Keyboard and Mouse Controls for the Amazon Sumerian Editor

The default camera that Sumerian adds to every scene supports mouse controls for pan, zoom, and orbiting around the camera's anchor point. To move the camera, press and hold a mouse button while you move the mouse. If you only have one mouse button, you can use a keyboard key plus mouse button combination to perform the same movements.

Camera Movement

- **Zoom in and out** – Mouse wheel scroll up and down
- **Pan** – Mouse wheel button, or SHIFT + left mouse button
- **Orbit** – Right mouse button, or ALT + left mouse button

The Sumerian editor provides keyboard equivalents of most of the canvas menu buttons (p. 13). Use the bottom row of keys to switch between preset camera views, and the **f** key to fill the canvas with a single entity. The space bar hides the side panels to let the canvas fill the screen.

Camera

- **Frame entity** – f
- **Frame all** – SHIFT + f
- **Bottom and top views** – v
- **Back and front views** – c
- **Left and right views** – x
- **Editor camera view** – z
- **Show and hide side bars** – space

Select entities by clicking on them in either the editor or the **Entities** panel. With an entity selected, use the following commands to speed up editing.

Editing

- **Delete entity** – backspace or delete
- **Duplicate entity** – CTRL + d
- **Translate handles** – w
- **Rotation handles** – e
- **Scale handles** – r
- **Switch between global and relative transform** – g
- **Undo** – CTRL + z
- **Redo** – CTRL + SHIFT + z

Use the following commands to open the text editor, timeline, and publishing menus.

Tools

- **Text editor** – j
- **Timeline** – t
- **Publish** – CTRL + SHIFT + p

With the timeline open, use the following commands to adjust keyframes and playheads.

Timeline

- **Move keyframe left** – left (fast), CTRL + left (slow)
- **Move keyframe right** – right (fast), CTRL + right (slow)

- **Move playhead left** – SHIFT + left (fast), CTRL + SHIFT + left (slow)
- **Move playhead right** – SHIFT + right (fast), CTRL + SHIFT + right (slow)
- **Align keyframe left** – CTRL + ALT + 1
- **Align keyframe center** – CTRL + ALT + 2
- **Align keyframe right** – CTRL + ALT + 3
- **Move keyframe to start** – home
- **Move keyframe to end** – end

Publishing Scenes in the Amazon Sumerian Editor

Publish your scene to Amazon CloudFront to share it with users.

To publish a scene

1. Open your scene in the editor.
2. Choose **Scene**, and then choose **Publish**.
3. Configure publishing settings.
    - **Custom CSS** – specify the contents of a `style` tag that you want to add to the generated web page.
    - **Custom JavaScript** – specify the contents of a `script` tag that you want to add to the generated web page.
4. Choose **Publish**.

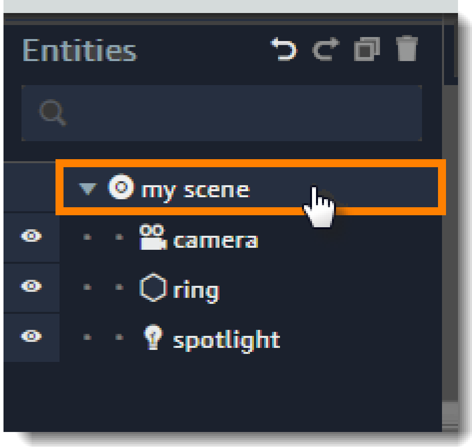
After a few seconds, your scene is live. You can click *View* to open it.

Amazon Sumerian Scene Settings

The editor contains many options for configuring a scene in addition to the options available in the dashboard (p. 8). In the inspector panel, you can configure credentials for the AWS SDK for JavaScript, adjust the canvas size and grid, and configure global settings like fog, background image, and post processing effects.

To configure a scene

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



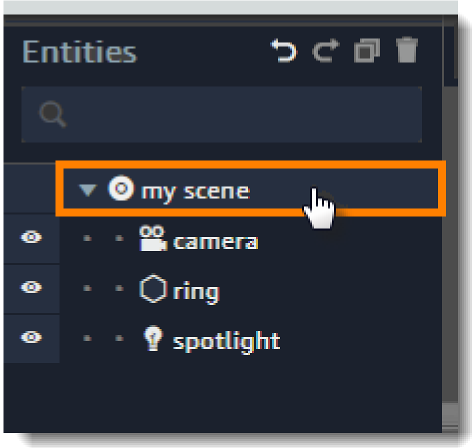
3. Modify scene settings in the inspector panel.

   - **Details** – Update the scene's name and description.
   - **Tags** – Add metadata tags to the scene.
   - **Custom attributes** – Add metadata key-value pairs to the scene.

You can save a copy of your scene as a **template** to use it as a starting point for creating other scenes. If your scene is saved to a project, the template is saved to the same project. Otherwise, you must choose a project to hold the template.

To create a template

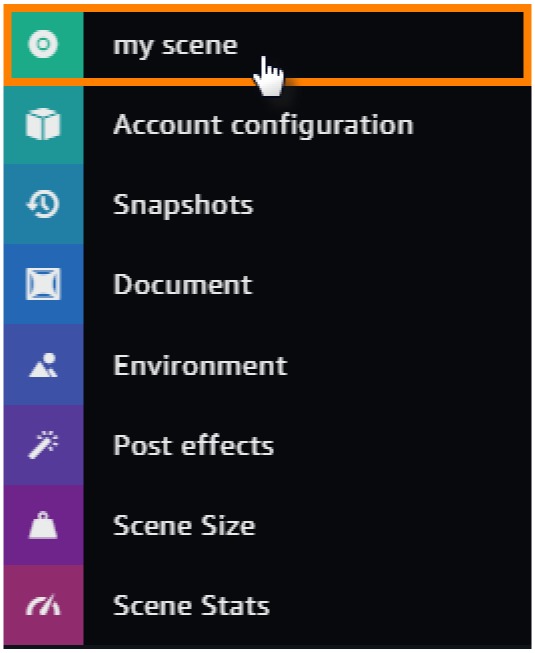
1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



3. Expand the scene section in the inspector panel.
4. Choose **Save scene as template**.
5. (drafts) Choose a project for the template.

You can copy or move templates between scenes (p. 10) in the dashboard. You can update a template by creating a template again from the same source scene or a scene created from the template. When you save a template, you can choose to create a new template or update the existing template.

Collapse the scene settings section by clicking on the name of your scene.



The following topics describe the settings available in each section.

Sections

- Configuring AWS Credentials for Your Amazon Sumerian Scene (p. 22)
- Taking Snapshots of Your Amazon Sumerian Scene (p. 23)
- Configuring the Canvas for Your Amazon Sumerian Scene (p. 25)
- Configuring Environment Settings for Your Amazon Sumerian Scene (p. 26)

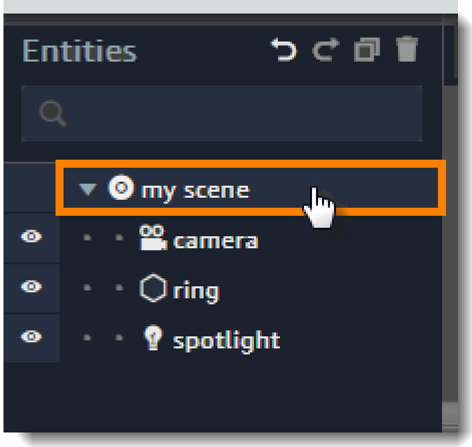
Configuring AWS Credentials for Your Amazon Sumerian Scene

The **AWS configuration** section lets you configure credentials for use with the AWS SDK for JavaScript. You can set a Amazon Cognito identity pool ID, which Sumerian will use to retrieve credentials when the scene is loaded. The identity pool must have an unauthenticated role with permission to use the AWS APIs that your scripts access.

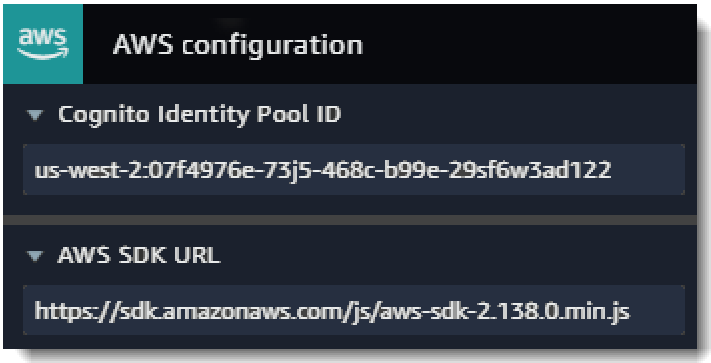
> **Note**
> If you don't have an identity pool, follow the instructions under Amazon Sumerian Permissions (p. 5) to create one.

To configure AWS SDK for JavaScript credentials

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



3. Expand the **AWS configuration** section in the inspector panel.
4. Enter an Amazon Cognito identity pool ID.



AWS configuration

Cognito Identity Pool ID

us-west-2:07f4976e-73j5-468c-b99e-29sf6w3ad122

AWS SDK URL

https://sdk.amazonaws.com/js/aws-sdk-2.138.0.min.js

To use the credentials, create a script that listens for `aws.sdkReady` before initializing an SDK for JavaScript client. The following example lists the contents of an Amazon S3 bucket named `mybucket` in the browser console.

Example S3listobjects

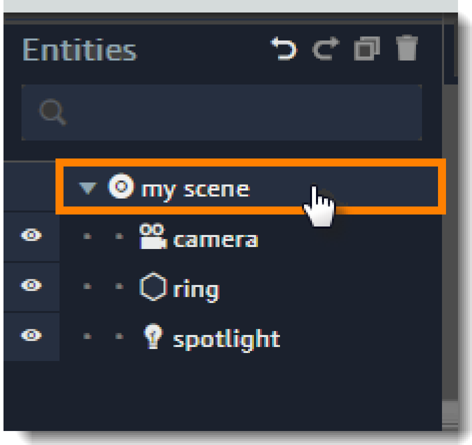
```
var setup = function(args, ctx) {
    sumerian.SystemBus.addListener('aws.sdkReady',
        () => {
            let s3 = new AWS.S3();
            s3.listObjects({Bucket: "mybucket"}, function(err, data) {
                    if (err) {
                        console.log('ERROR', err, data);
                    } else {
                        console.log('DATA', data);
                    }
                }
            );
        },
        true
    );
};
```

Taking Snapshots of Your Amazon Sumerian Scene

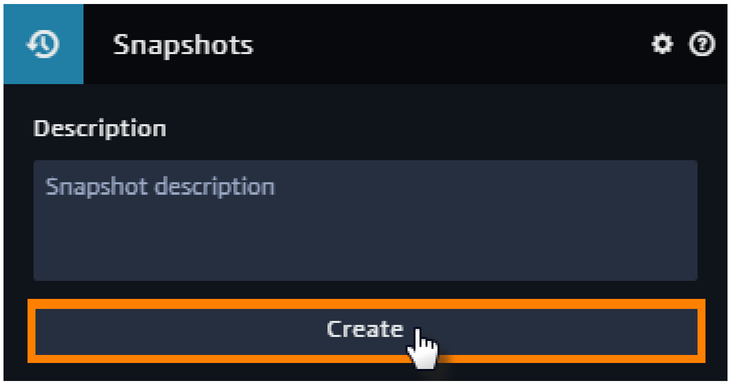
The Snapshots panel lets you create a copy of your scene that you can restore later. This is useful when doing experimental changes to the scene.

To create a snapshot

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



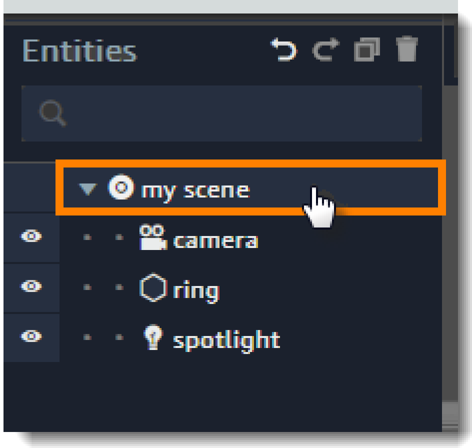
3. Expand the **Snapshots** section in the inspector panel.



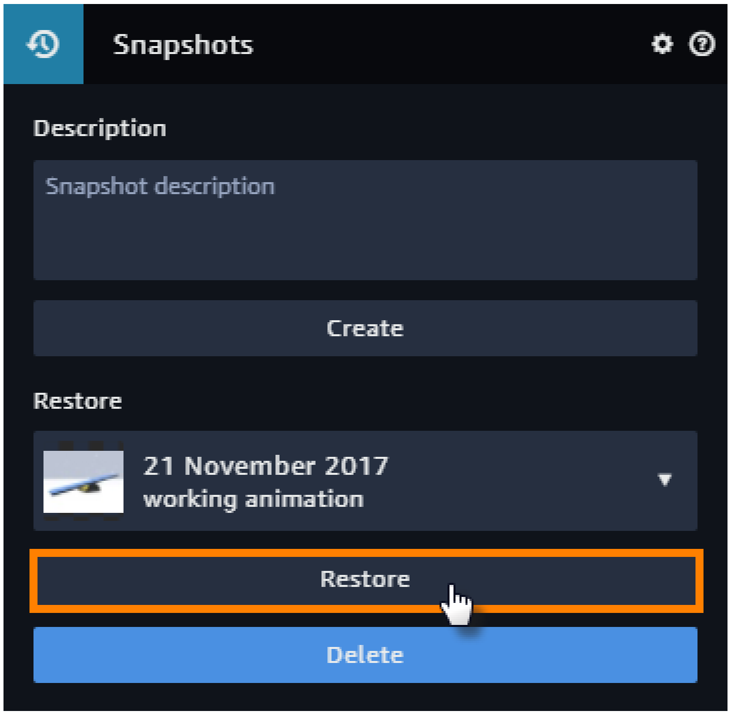
4. Enter a description.
5. Choose **Create**.

To restore or delete a snapshot

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



3. Expand the **Snapshots** section in the inspector panel.
4. Choose a snapshot.
5. Choose **Restore** or **Delete**.

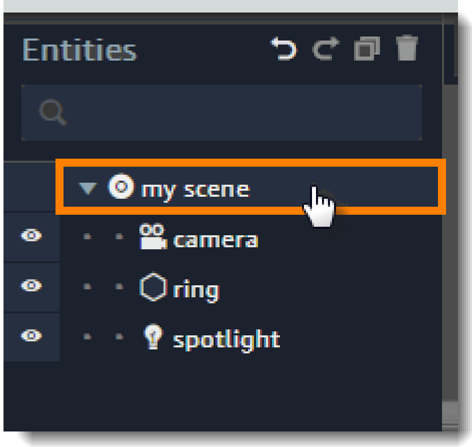


Configuring the Canvas for Your Amazon Sumerian Scene

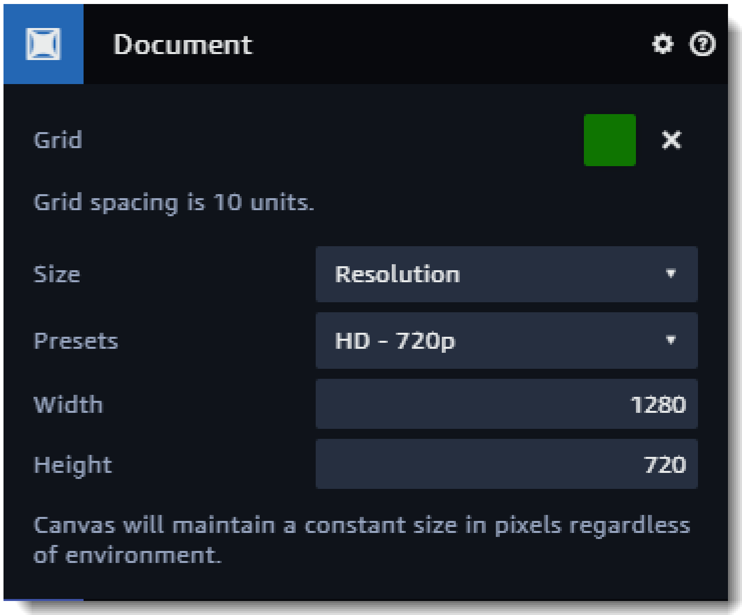
The document panel lets you configure the size of the WebGL canvas and grid color. These settings only apply both while you are working on a scene in the editor, and to the published scene

To configure document settings

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



3. Expand the **Document** section in the inspector panel.



Document

Grid 🟩 ✕

Grid spacing is 10 units.

Size	Resolution
Presets	HD - 720p
Width	1280
Height	720

Canvas will maintain a constant size in pixels regardless of environment.

4. Choose from the following options.

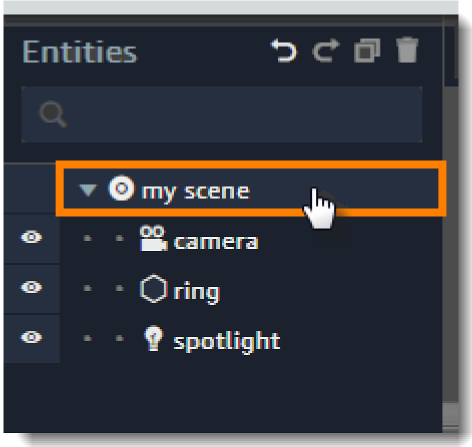
   - **Grid** – change the color of the grid.
   - **Stretch** – stretch the canvas to its container.
   - **Aspect ratio** – stretch the canvas to its container, but keep the aspect ratio.
   - **Resolution** – set a fixed size of the canvas.

Configuring Environment Settings for Your Amazon Sumerian Scene

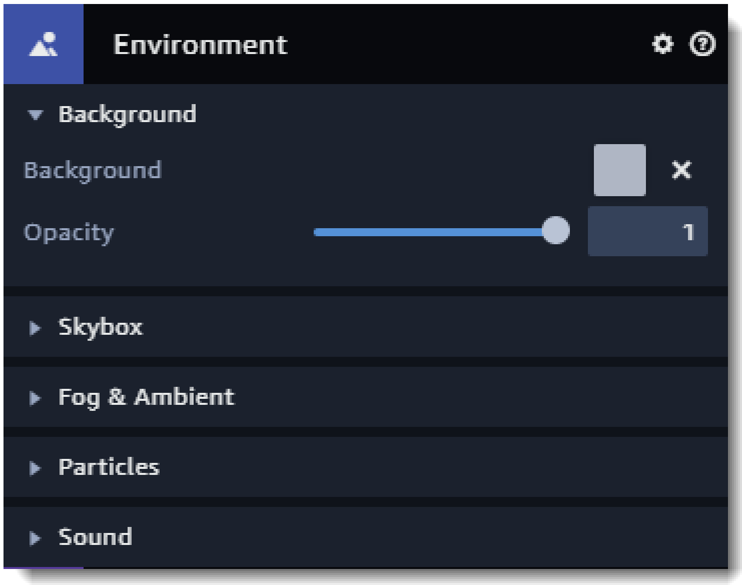
Use environment settings to configure your scene's background image, ambient lighting, and weather.

To configure environment settings

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



3. Expand the **Environment** section in the inspector panel.



Environment

Background

Background

Opacity 1

Skybox

Fog & Ambient

Particles

Sound

4. Configure the following settings.

   - **Background** – set the background color of the scene, and its opacity.

     To make the background transparent, set **Opacity** to `0`. Background settings have no effect if you add a **Skybox**.
   - **Skybox** – use an image as the background of the scene. You can drop an existing skybox from the assets panel, or choose the plus icon to create a new skybox (p. 55).
   - **Ambient** – add ambient light to light all objects the scene. Ambient light does not affect the skybox.
   - **Fog** – add fog to the scene. Fog starts occluding objects in the scene at **Fog near** units from the camera, and strengthens until **Fog far** units, where it becomes completely opaque.
   - **Particles** – add animated snow-like particles to the background of the scene.

     Properties

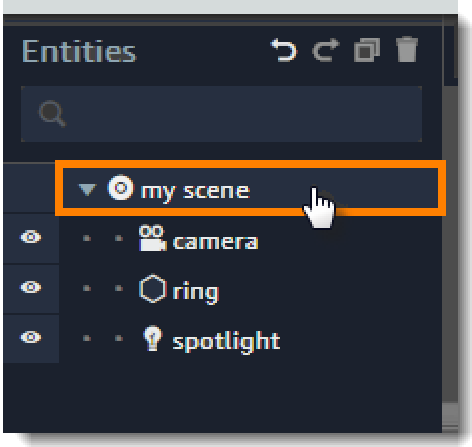
     - **Velocity** – the speed of the falling particles.
     - **Rate** – the number of particles that appear per second.
     - **Height** – the height at which the particles will appear, relative to the camera height.

Configuring Post Processing Effects for Your Amazon Sumerian Scene

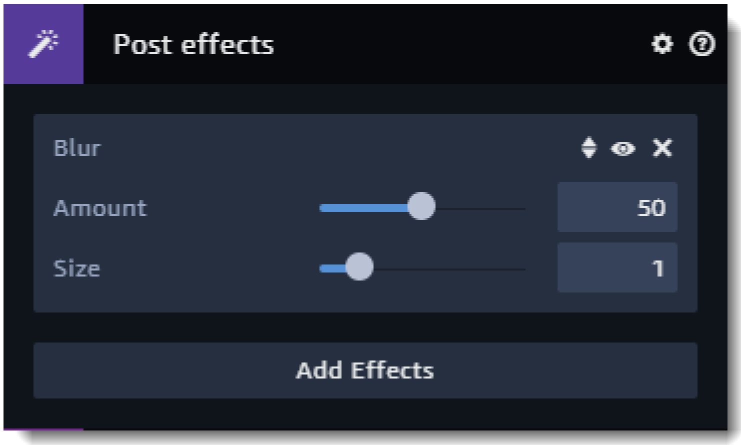
In the post effects panel it is possible to create a stack of effects, affecting the final render composition. Post effect layers applied as render passes by the Sumerian engine.

To add post effects

1. Open a scene in the Sumerian editor.
2. Choose the root node in the **Entities** panel.



3. Expand the **Post effects** section in the inspector panel.



Blur

Amount 50

Size 1

4. Choose **Add effects**.
5. Choose one or more effects and then choose **Add**.
6. Adjust the settings for each effect in the inspector panel.
7. See how post effects affect rendering by clicking the post effects icon in the canvas toolbar to toggle them on and off.

Post effect properties

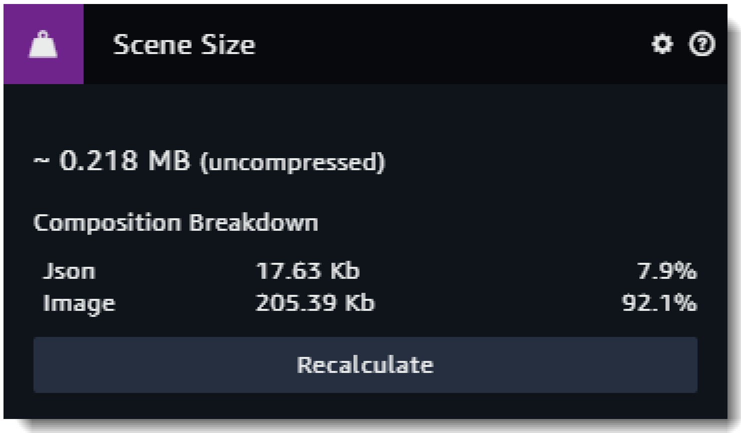
- **Antialias** – add FXAA based antialiasing to smooth out jagged edges.
  - **Span** – the area of the smoothing effect.
- **Bloom** – intensify and create glow out of the high valued colors in the input.
  - **Opacity** – the amount of bloom applied.
  - **Size** – the size of the glow area.
  - **Gain** – the amount of brightness added.
  - **Intensity** – contrast.
- **Bleach** – alter input color by its luminance.
  - **Opacity** – the blending multiplier for the effect.
- **Blur** – blur the entire scene to make it appear out of focus.

  - **Amount** – the amount of blending.
  - **Size** – the size of the blur area.
- **Contrast** – adjust the brightness, contrast and saturation.
  - **Brightness** – remove or add brightness.
  - **Contrast** – adjust the contrast.
  - **Saturation** – adjust the color saturation.
- **Dot** – add a black and white lattice effect.
  - **Angle** – the angle of the lattice.
  - **Scale** – the thickness of the lattice.
  - **SizeX** – skew the lattice on the X axis.
  - **SizeY** – skew the lattice on the Y axis.
- **Edge detect** – add a *difference of Gaussians*-based edge detection.
  - **Gauss Sigma** – the base of the two Gaussian kernels.
  - **Threshold** – the edge detection tolerance value.
  - **Background %** – the amount of blending between the background and edge colors.
  - **Edge Color** – the edge color.
  - **Background Color** – the background color.
- **Film grain** – add noise and resolution lines.
  - **Noise** – the amount of noise.
  - **Line Intensity** – the sharpness of the lines.
  - **Line Count** – the number of lines.
- **Hatch** – render the scene in black and white, with a lattice effect over black areas.
  - **Width** – the width of the lattice lines.
  - **Spread** – the distance between the lattice lines.
- **HSB** (hue, saturation, and brightness) – adjust colors of the scene.
  - **Hue** – adjust the hue.
  - **Saturation** – adjust the color saturation.
  - **Brightness** – adjust the image brightness.
- **Levels** – apply gamma correction to the image.
  - **Gamma** – adjust the gamma level.
  - **Min input** and **Max input** – the gamma input range.
  - **Min output** and **Max output** – the gamma output range.
- **Motion Blur** – apply a blur effect to objects that moved since the previous rendered frame. If the camera moves, the entire image blurs.
  - **Amount** – the amount of blending.
  - **Scale** – overlay the previous frame on top of the current frame at a different scale to create a zooming or flying effect.
- **Noise** – add signal noise to the image.
  - **Noise** – the amount of noise.
- **Overlay** – overlay a texture on the image.
  - **Texture** – the texture asset.
  - **Blend mode** – the method of blending the overlay and background.
  - **Amount** – the amount of blending.
- **Radial** – add a radial blur to the image.
  - **Offset** – the blur offset.
  - **Multiplier** – the blur multiplier.

- **RGB shift** – split the image into red, green, and blue layers with an offset between layers.
  - **Amount** – the distance between the layers.
  - **Angle** – the angle in radians between the layers.
- **Sepia** – add a sepia color filter.
  - **Amount** – the intensity of the effect.
- **Tint** – apply a color filter to the image.
  - **Color** – the tint color.
  - **Amount** – the intensity of the effect.
- **Vignette** – add a dark gradient around the edges of the image.
  - **Offset** – the size of the gradient.
  - **Darkness** – the strength of the gradient.

Calculating the Size of Your Amazon Sumerian Scene

The **Scene Size** section calculates how much data your scene is using. You can see the number of kilobytes from JSON, mesh data and binaries.



Scene Size

~ 0.218 MB (uncompressed)

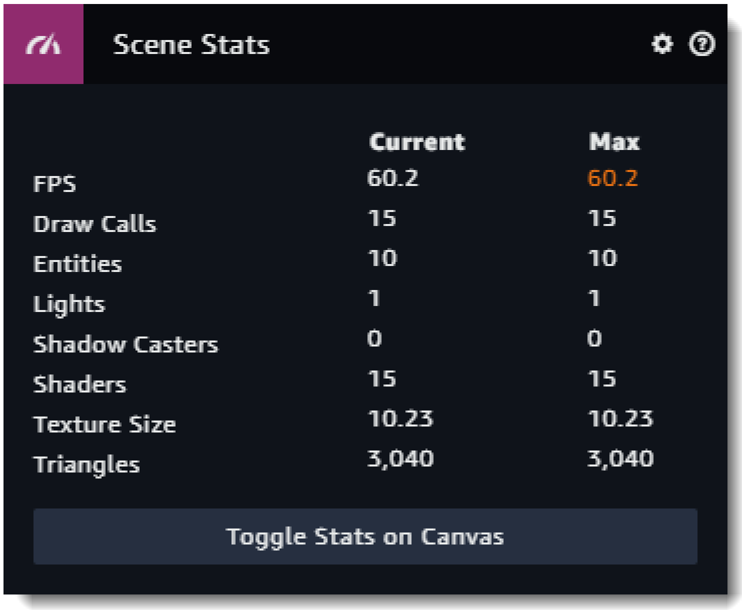
Composition Breakdown

Json	17.63 Kb	7.9%
Image	205.39 Kb	92.1%

The numbers shown reflect the size of the scene uncompressed. When a scene is served from Amazon CloudFront, the contents are compressed. To see the compressed size, open your scene in a browser and use developer tools to find the amount of data transferred.

Viewing Performance Information for Your Amazon Sumerian Scene

The **Scene stats** section gives you some stats about the current scene relating to performance.



	Current	Max
FPS	60.2	60.2
Draw Calls	15	15
Entities	10	10
Lights	1	1
Shadow Casters	0	0
Shaders	15	15
Texture Size	10.23	10.23
Triangles	3,040	3,040

Stats

- **FPS** – the number of frames rendered per second.
- **Draw calls** – the number of draw calls made per frame.
- **Entities** – the number of visible entities.
- **Lights** – the number of lights being rendered.
- **Shadow casters** – the number of shadow casters being rendered.
- **Shaders** – the number of shaders being rendered.
- **Texture size** – the current texture size in the GPU.
- **Triangles** – the number of triangles being rendered.

To improve performance, try reducing the number of lights, shaders, and draw calls.

Entities

An entity is a container of components. Each component adds some functionality to an entity. For example, when a transform component is added to an entity, the entity has a position in the 3D world. If a camera component is added to the same entity, we now have a camera with a position. If a Geometry component and a Material component are added to an entity, it's now a 3D model we can render.

To create an entity and add components

1. Open a scene in the Sumerian editor.
2. Choose **Create entity**.
3. Choose a shape or built-in object. For a blank entity, choose **Entity**.
4. Choose the new entity in the entities panel.
5. In the inspector panel, expand the **Details** section. Enter text in the fields to change the entity name, description, tags, and attributes.
6. To place the entity in the scene, use the X, Y and Z axis handles in the canvas, or expand the **Transform** section and enter absolute coordinates.
7. Choose the plus icon, and then choose a component type to add a component to the entity.

All entities include a transform component that you can use to move the entity around the scene. Depending on the type of entity that you create or import, other components are also included automatically.

Default Components

- **2d shapes** – transform, 2d graphics
- **3d shapes** – transform, geometry, material
- **Cameras** – transform, camera, script
- **Host** – transform, host, speech
- **HTML 3D** – transform, HTML 3D
- **Lights** – transform, light
- **Particles** – transform, particles
- **Timeline** – transform, timeline

Use the entities panel to organize your scene's entities in a hierarchy. When you make an entity a child of another entity, it attaches to the parent. That is, when you move the parent, the child moves, and when you set the child's position, it is relative to the position of the parent.

To manage entities

1. Open a scene in the Sumerian editor.
2. Choose an entity in the entities panel.
3. Drag the entity onto another entity to make it a child of that entity.
4. Click the eye icon next to an entity to hide it.
5. Click the copy icon to duplicate it.

6. Click the trash icon to delete it.

Some components also have special properties that are affected by the components on their parent. For example, a collider's behavior changed depending on the type of rigid body attached to the same entity or parent entity.

To manage components

1. Open a scene in the Sumerian editor.
2. Choose an entity in the entities panel.
3. In the inspector panel, click a component's name to expand or collapse its properties. See the topic for each component for details on the available properties.
4. Click the cog icon and use the following options.
    - **Reset** – restore the default values for the component's properties.
    - **Toggle panel** – show or hide the component properties.
    - **Remove** – delete the component.
    - **Copy** (some components) – copy the component configuration.
    - **Paste** (some components) – paste the component configuration copied from the same component on a different entity.

The following topics describe the parameters and usage of each type of component.

Components

The Amazon Sumerian Transform Component

The transform component contains the local transform of the component –translation, rotation and scale. The transform is relative to its parent.

Properties

- **Translation** – the position of the object relative to its parent.
- **Rotation** – the rotation of the object in degrees.
- **Scale** – the size of the object.
- **Uniform scale** – maintain proportions when scale is modified on any axis.
- **Static** – optimize the object's rendering based on it not moving during playback.

The Amazon Sumerian Geometry Component

The *Geometry* panel contains a renderable mesh or primitive on the entity. When you import a 3D model, or create a primitive from the *Create Entity* dialog, it will always get a Geometry component.

Together with a Material component, the Geometry can be rendered. If you don't have a Material component, the geometry will be invisible.

The panel looks different for different kinds of meshes and primitives, but in general, they have these settings:

Properties

- Cast shadows
- Receive shadows
- Dimensions and/or number of samples (primitives only)

The Amazon Sumerian Material Component

When you add a 3D model to your environment, it has at least two components. The geometry (p. 34) component defines the shape of the model. The material component defines its textures and rendering properties.

Properties

- **Diffuse color** – the base color of the surface.
  - **Color** – the base diffuse color.
  - **Texture** – set the diffuse color from a texture.
- **Normal** – normal maps are a type of Bump Map. They are a special kind of texture that allow you to add surface detail such as bumps, grooves, and scratches to a model which catch the light as if they are represented by real geometry.

  You set the normal map via the **Texture** input, and you can alter its magnitude by setting the **Strength** value.
- **Specular** – specular effects are essentially the direct reflections of light sources in your scene which typically show up as bright highlights or shines on the surface of objects (although specular highlights can be subtle or diffuse too).

  You can set the base specularity **Color**, use a **Texture** and set the **shininess** value.
- **Emissive** – this is the self-illumination color an object has. You can set the emissive color using the **Color** input and/or via a **Texture**.
- **Ambient** – ambient color is the color of an object where it is in shadow. This color is what the object reflects when illuminated by ambient light rather than direct light.

  - **Color** sets the base ambient color.
  - **Texture** this ambient map identifies areas on a mesh that are exposed or hidden from ambient lighting.
- **Opacity** – the opacity is used when using the *Transparent Blending* mode. The **Strength** allows you to input a value between 0 and 1 where 0.0 represents completely transparent and 1.0 represents fully opaque.
  - **Threshold** is used to indicate when a surface is completely transparent, and can be discarded from rendering.
  - **Dual Transparency**
- **Reflectivity**
  - **Texture** – reflectivity texture.
  - **Environment** – environment map that you will see in the reflection. If not selected, the current Skybox will be used.
  - **Amount** – amount of reflectivity to use.
  - **Fresnel** – a nonzero fresnel value will result in less reflection depending on the normal direction.
- **Refractivity** – the Refraction input takes in a texture or value that simulates the index of refraction of the surface. This is useful for things like glass and water, which refract light that passes through them. The environment texture will be used for the refraction.
  - **Amount** – how much refraction to blend with the current color.
  - **Refraction** – the ratio of the refractive indices involved in the refraction.
- **Blending** – the blending mode to use for the material.
  - **NoBlending**
  - **TransparencyBlending**
  - **CustomBlending**
  - **AdditiveBlending**
  - **SubtractiveBlending**
  - **MultiplyBlending**
- **Culling** – whether to cull on the triangle level, and which face (back, front, both) to cull.
- **Depth** – whether to enable depth testing, depth writing and which **RenderQueue** value to use.
- **Shading**
  - **Flat** – turns on flat shading for the mesh.
  - **Wireframe** – renders the mesh in wireframe mode.
  - **Wrap factor** – the wrap factor.
  - **Wrap amount** – the wrap amount.

Note that Material Assets can be shared between entities. If a Material Asset is shared, changing the look of one entity will also change the ones which are sharing the material.

The Amazon Sumerian Camera Component

The camera component adds a camera to your entity. In the editor, you can use it to define your 3D viewport.

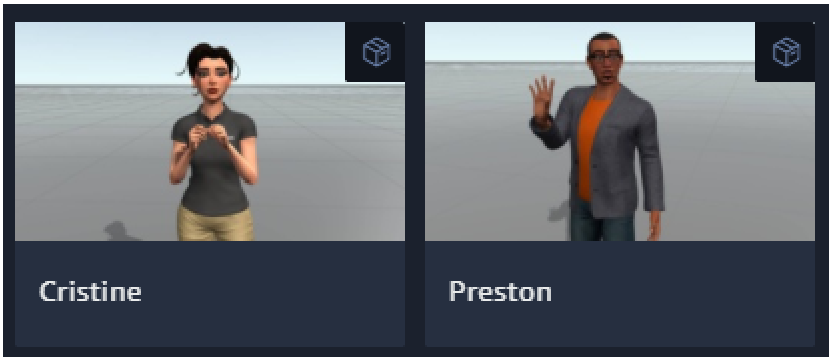
Properties

- **Main camera** – use this camera at the beginning of the scene. If you have multiple cameras, you can switch between these during both edit- and playback mode.

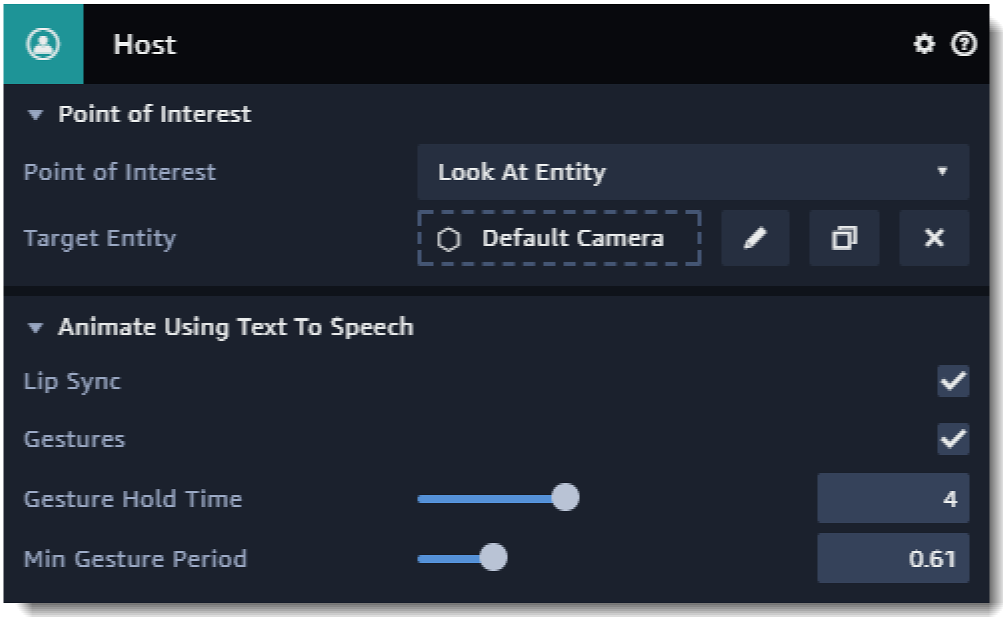
- **Follow editor camera** – set the camera position with the editor camera. This is useful when switching between play and edit mode.
- **Projection** – This setting controls how the camera will project the 3D world on the 2D canvas.
- **Field Of View (FOV)** – the number of degrees from left to right that the camera will span.
- **Clipping planes** – the distance from the camera at which objects are drawn.

The Amazon Sumerian Host Component

A host is a asset provided by Sumerian that has built in animation, speech, and behavior for interacting with users. Add a host to your scene from the asset library (p. 14).



When you add a host to your scene, it includes a **Host** component for configuring the host's behavior, and a **Speech** component (p. 37) that you can use to configure the host's voice and script.


Host

Point of Interest

Point of Interest: Look At Entity

Target Entity: Default Camera

Animate Using Text To Speech

Lip Sync ☑

Gestures ☑

Gesture Hold Time: 4

Min Gesture Period: 0.61

Properties

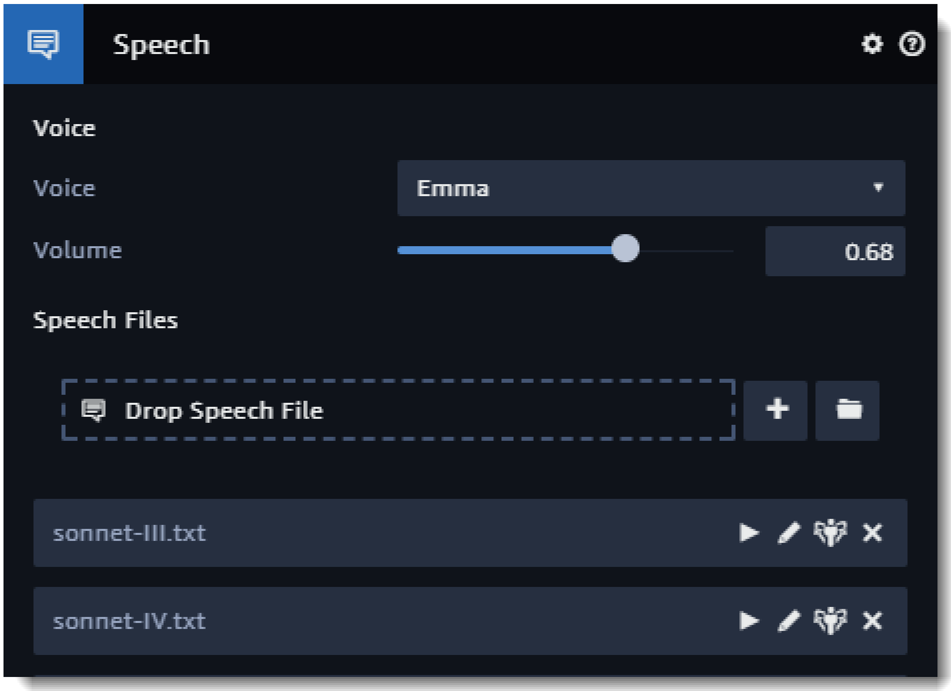
- **Point of interest** – set to **Look at entity** to keep the host's eyes trained on a camera, object, or other entity during playback.
- **Target entity** – drop an entity here to set it as the host's point of interest.
- **Lip sync** – play lip sync animations during speech.

- **Gestures** – play gesture animations during speech.
- **Gesture hold time** – the number of seconds to play a gesture animation.
- **Minimum gesture period** – number of seconds to wait after a gesture is complete before another gesture can occur.

The Amazon Sumerian Speech Component

The speech component assigns text to an entity for playback with Amazon Polly. You assign text to an entity, and play the audio output from Amazon Polly with a state machine or script. The scene calls Amazon Polly at runtime to generate the audio.

To use Amazon Polly during playback, the scene needs AWS credentials from Amazon Cognito Identity. Create an identity pool (p. 5) for your scene, and configure it under AWS configuration (p. 22) in the scene settings.



Speech

Voice

Voice: Emma

Volume: 0.68

Speech Files

sonnet-III.txt

sonnet-IV.txt

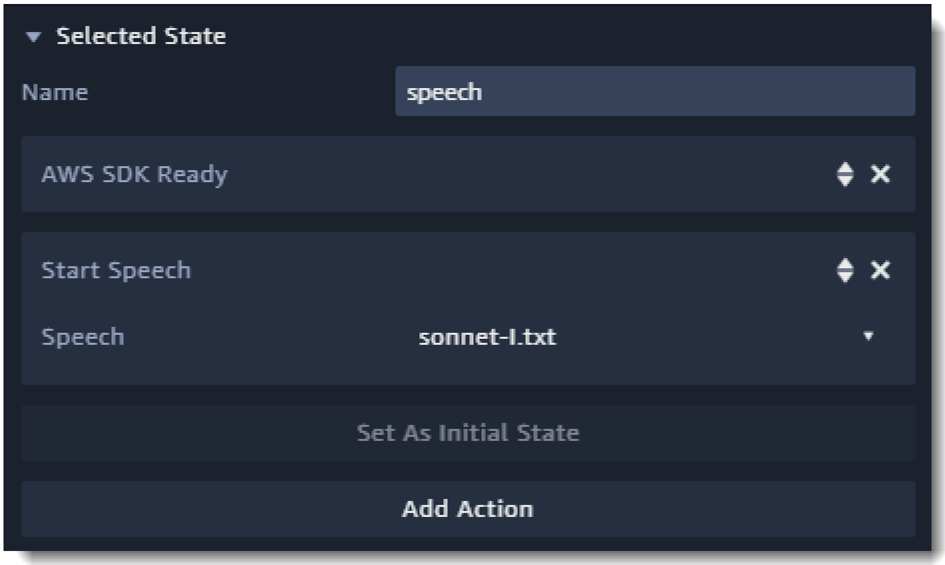
Properties

- **Voice** – an Amazon Polly voice.
- **Volume** – volume of the rendered audio.
- **Speech files** – drop text files here to add them to the component. Click to mark up a speech file with gestures.

To trigger a speech during playback, use a state machine or script component on the same entity.

State Machine

To play a speech, add a state machine (p. 45) to the entity with the speech component. Add a state with **AWS SDK ready** and **Start speech** actions.



▾ Selected State

Name: speech

AWS SDK Ready

Start Speech

Speech: sonnet-I.txt

Script

To play a speech, get a reference to the speech component from the context object. The component has a `speeches` array that contains the speeches attached to the component. Call `play` on a speech.

Sumerian calls Amazon Polly when you play a speech, so you must use the `aws.sdkReady` listener to ensure that your scene's AWS credentials are loaded prior to the call.

Example script – play a random speech

```
'use strict';
var setup = function(args, ctx) {
  sumerian.SystemBus.addListener('aws.sdkReady',
    () => {
      var speechComponent = ctx.entity.getComponent("speechComponent");
      var speeches = speechComponent.speeches;
      var speech = speeches[Math.floor(Math.random() * speeches.length)];
      speech.play();
    },
    true
  );
};
```

The Amazon Sumerian 2D Graphics Component

A 2 dimensional image or video.

Properties

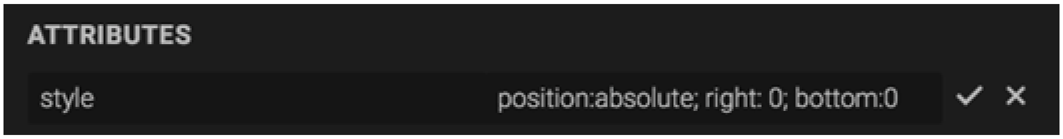
- **Tint** – the tint color.
- **Emissiveness** – the emissiveness of the image.
- **Opacity** – the opacity of the image.
- **Reflection** – the reflectivity of the image.

The Amazon Sumerian HTML Component

The HTML component adds a 2D DOM element to the entity, and lets you edit its HTML contents using the text editor. You can position the component using the entity transform, or use CSS to position it relative to the viewport.

Properties

- **Move with transform** – uncheck to position the component with CSS, instead of using the transform component.
- **Pixel perfect** – snap the HTML component to the closest pixel position.
- **Attributes** – add HTML attributes to the DOM. This can be useful for positioning, for example. If you want to set the position to the bottom right corner, you can do this:



style position:absolute; right: 0; bottom:0

  Note that other attributes on the DOM element will be overridden by the attributes you add here.

Choose the **Open in editor** button to open the HTML editor for the component. This lets you edit the HTML contents.

This is the HTML used for the HTML component shown in the screenshot above.

```
<style>
.my-paragraph{
    background:#fefefe;
    font-size:16px;
    padding:10px;
    border-radius:3px;
    margin:0;
    font-family:sans-serif;
}
</style>
<pclass="my-paragraph">
    This is HTML
</p>
```

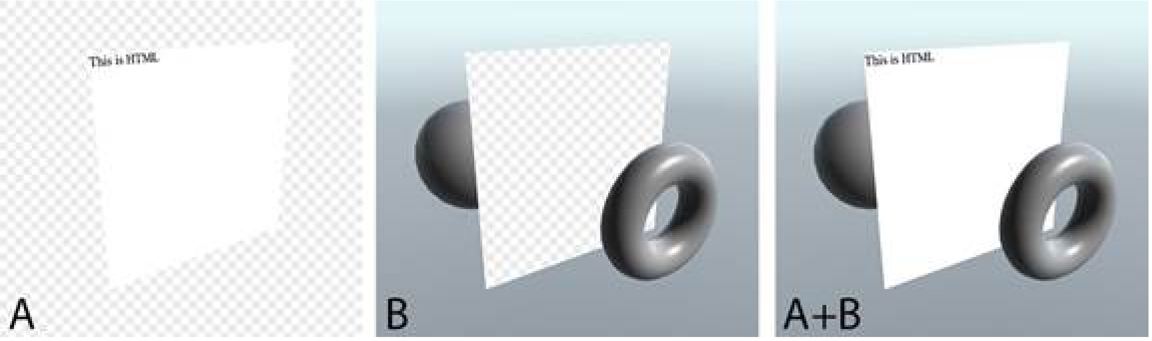
The Amazon Sumerian HTML 3D Component

The HTML 3D component adds a 3D DOM element to the entity, and lets you mix it with WebGL easily. The 3D DOM element behaves like a Quad in 3D space and you can edit its content in the text editor.

Properties

- **Width** – The number of pixels that should fit into the 3D quad along the X axis.

The technique behind the HTML 3D component is as follows. The WebGL canvas is put in front of a DOM element, which is transformed using CSS3D. Inside the WebGL scene, there's a Quad that masks the Canvas, so that the DOM element behind the canvas is visible through. Synchronization of the WebGL and CSS3D transforms is done by the Sumerian engine internally.



The most prominent limitation is that you cannot use transparency in your HTML. Since there's nothing behind the 3D DOM element, you will see empty background behind it.

To embed an iframe, simply use the following HTML for your HTML 3D component (just replace the URL).

```
<iframe src="https://en.wikipedia.org/wiki/WebGL" width="100%" height="100%"></iframe>
```

Embed a YouTube video

Go to a YouTube video, click *Share*, then *Embed*, and copy the embed code. Paste it into your HTML3D component.

```
<iframe width="100%" height="100%" src="https://www.youtube.com/embed/qpQFfMofc1I"
 frameborder="0" allowfullscreen></iframe>
```

The Amazon Sumerian Sound Component

The sound component adds a number of sound assets to the entity. The sound will not auto-play, but you can play it using a script or the state machine.

Properties

- **Master volume** – The volume of all sounds in the component.

The Amazon Sumerian Light Component

The light component adds a light source to the entity.

Properties

- **Type**
  - **Point** – emits light in all directions from a point in space, like a flame.
  - **Directional** – emits light uniformly over the entire scene, like the sun.
  - **Spot** – emits light in a cone, like a spotlight.
- **Color** – the color of the light.
- **Intensity** – the intensity of the light (typically between 0 and 1).
- **Specular** – the intensity of the specular light (typically between 0 and 1).
- **Range** (point and spot) –

- **Cone angle** (spot) – the angle of the cone at the light source, in degrees.
- **Penumbra** (spot) – the intensity of the light near the edges of the cone.
- **Projection** (directional and spot) – upload a texture to apply to the light.
- **Shadows** (directional and spot) – cast shadows from objects that the light hits.

The Amazon Sumerian Particle System Component

The particle system component simulates fluid entities such as liquids, clouds and flames by generating and animating large numbers of small 2D images in the scene.

Properties

- **General** – The basic behavior of the particle emitter.
  - **Auto play** – Start the emission animation when the scene starts.
  - **Loop** – Loop the animation.
  - **Duration** – Duration of the animation in seconds.
  - **Prewarm** – Load the effect prior to playback.
  - **Max particles** – Limit the number of visible particles.
  - **Gravity** – Vector of the gravity force that applies to particles.
  - **Seed** – Randomization seed. Experiment with values to find a look that you like, or set to `-1` to get a different effect each time.
  - **Local space simulation** – Set to true to simulate the particle system within the boundaries of the parent entity, instead of in the entire scene.
- **Emitter shape** – The shape and size of the emitter. Additional settings are specific to each shape.

  #### Box

  - **Random direction** – emit each particle in a random direction.
  - **Box extents** – the height, width and length of the emitter.

  #### Sphere

  - **Radius** – the size of the emitter.
  - **Emit from shell** – emit particles from the outside edge of the emitter.
  - **Random direction** – emit each particle in a random direction.

  #### Cone

  - **Random direction** – emit each particle in a random direction.
  - **Emit from** – emit particles from the narrow end of the cone (**Base**), the center of the cone (**Volume**), or the edges of the cone (**Volumeedge**).
  - **Cone radius** – the radius of the cone at the narrow end.
  - **Cone angle** – the angle at which the sides of the cone flare out.
  - **Cone length** – the length of the sides of the cone.
- **Over duration properties** – fine tune values that apply to each loop of animation. Each value can be constant, or progress linearly or randomly over the duration.
  - **Emission rate** – the number of particles emitted per second.
  - **Start speed** – the speed of the particles.
  - **Start size** – the size of the particles.
  - **Start color** – the color of the particles.
  - **Start life time** – the number of seconds before each particle disappears.

  - **Start angle** – the angle of particles.
- **Over lifetime properties** – fine tune values that apply to the entire lifetime of the particle emitter. Each value can be constant, or progress linearly or randomly over the lifetime.
  - **Color** – the color of the particles. Compounds with the duration color.
  - **Size** – the size of the particles. Compounds with the duration size.
  - **Rotation speed** – rotation of particles in degrees per second.
  - **Local velocity** – local space velocity in units per second.
  - **World velocity** – world space velocity in units per second.
- **Texture** – the texture of each particle. Use one of the provided textures or choose **custom** to upload a texture.
- **Texture animation**
  - **Texture tiles** – the number of tiles in the sprite sheet, in X and Y directions.
  - **Cycles** – the number of exture animation cycles to finish over the lifetime.
  - **Frame over lifetime** – a curve specifying when to show which frame in the animation. `0` is the first frame and `1` is the last. A linear curve starting at 0 and ending at 1 traverses all frames in the animation.
- **Rendering** – customize the rendering behavior.
  - **Billboard** – particles always face the camera.
  - **Render queue** – render queue of the particle mesh.
  - **Render queue offset** – offset added to the render queue.
  - **Blending** – the type of blending (**None**, **Additive**, **Subtractive**, **Multiply**, or **Transparency**).
  - **Depth write** – write to the depth buffer.
  - **Depth test** – test against the depth buffer.
  - **Sorting mode** – the draw order for particles (**None** or **Camera distance**). For transparency blending, camera distance sorting is recommended.
  - **Opacity threshold** – The lower alpha threshold at which fragments are discarded.

The Amazon Sumerian Animation Component

The animation component controls the animations of an imported 3D mesh. It contains a list of *animation states* and *transitions*.

When you import your model into the editor, you get a geometry component as well as an animation component.

The animation **State** contains information about an animation, such as how many times it should loop, how fast it should run, and if it has any transition. If there's no transition for an AnimationState, then the Default Transitions at the bottom will be used.

If you want to switch between animation states, but want the transition between them to be smooth, then you add a **Transition**.

Transitions

- **Fade** – a transition that blends over a given time from one animation state to another, beginning the target clip from local time 0 at the start of the transition. This is best used with two clips that have similar motions.
- **SyncFade** – a transition that blends over a given time from one animation state to another, synchronizing the target state to the initial state's start time. This is best used with two clips that have similar motions.

- **Frozen** – a two state transition that freezes the starting state at its current position and blends that over time with a target state. The target state moves forward in time during the blend as normal.

The Amazon Sumerian Collider Component

The *collider component* adds collision geometry to the entity. If used together with a rigid body component, you can create a dynamic, colliding entity. If the collider doesn't have any rigid body component, it will become a static collision geometry in the physics world. We call this a *static collider*.

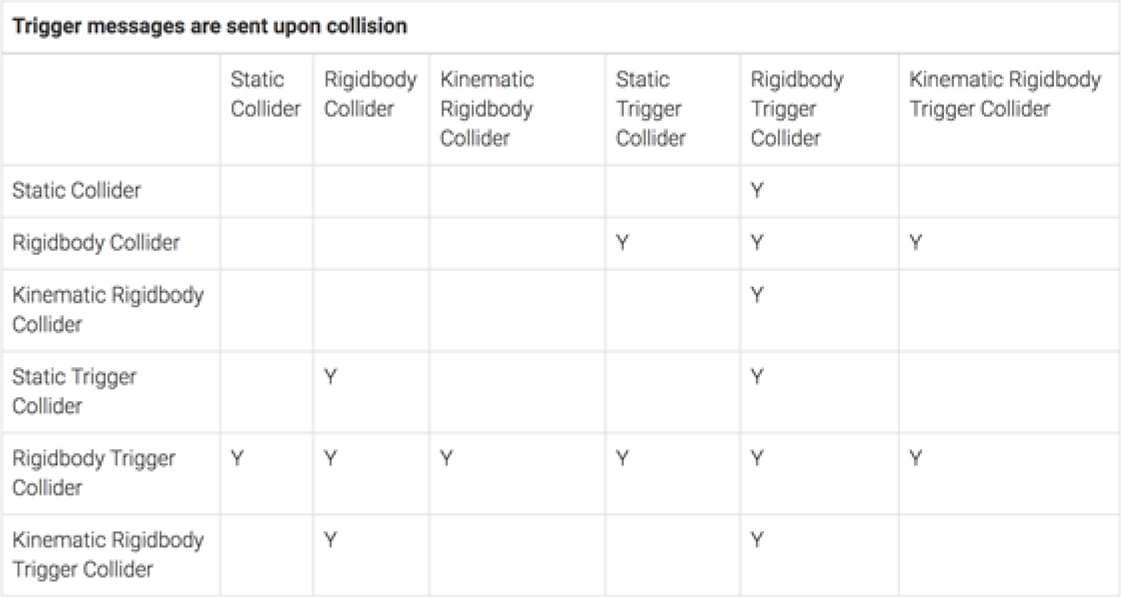
If the entity with a collider or any of its parents has a dynamic rigid body component, it will turn into a *dynamic collider*. If the entity with a collider or any of its parents has a kinematic rigid body component, it will turn into a *kinematic collider*.

The collider shapes are rendered with a green wireframe.

Properties

- **Shape** – The shape of the collider.
  - Box
  - Sphere
  - Plane
  - Infinite plane
- **Trigger** –

  If the collider is not a trigger, it will emit these events during collisions:
  - sumerian.physics.beginContact
  - sumerian.physics.duringContact
  - sumerian.physics.endContact

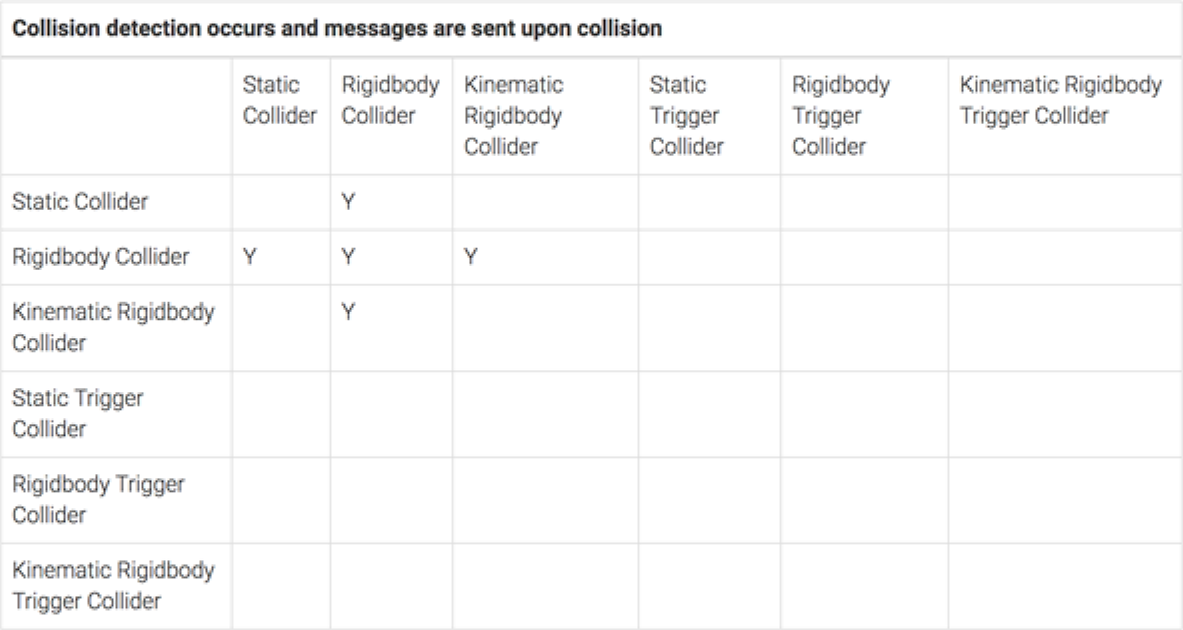


Trigger messages are sent upon collision

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					Y	
Rigidbody Collider				Y	Y	Y
Kinematic Rigidbody Collider					Y	
Static Trigger Collider		Y			Y	
Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y
Kinematic Rigidbody Trigger Collider		Y			Y	

  If the collider is a trigger, then it will *not* collide with other physics objects. However, it will emit events when a physics object enters it. Available events are:
  - sumerian.physics.triggerEnter

- sumerian.physics.triggerStay
- sumerian.physics.triggerLeave



Collision detection occurs and messages are sent upon collision

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		Y				
Rigidbody Collider	Y	Y	Y			
Kinematic Rigidbody Collider		Y				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

- **Friction** – `0` means no friction. The final friction (and restitution) value used in a collision is computed using multiplication. For example, a sphere with friction=0.5 that collides with a plane with friction=0.5 will get a friction value of 0.25.
- **Restitution** – how much the collider should bounce. `0` is no bounce and `1` is maximum bounce. If you set restitution to a number larger than one, it will gain more and more energy for each bounce.
- **Half extents** – the collider's half extents on the x, y, and z axises.

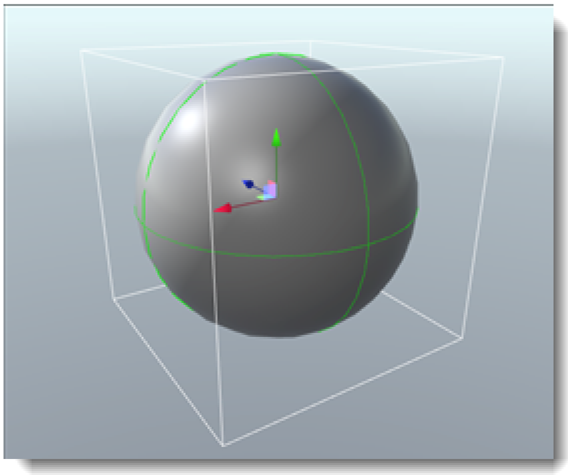
The Amazon Sumerian Rigid Body Component

The rigid body component adds physics properties, such as mass and velocity, to the entity. The component will simulate physics for the component and set the position and orientation of the entity accordingly.

Properties

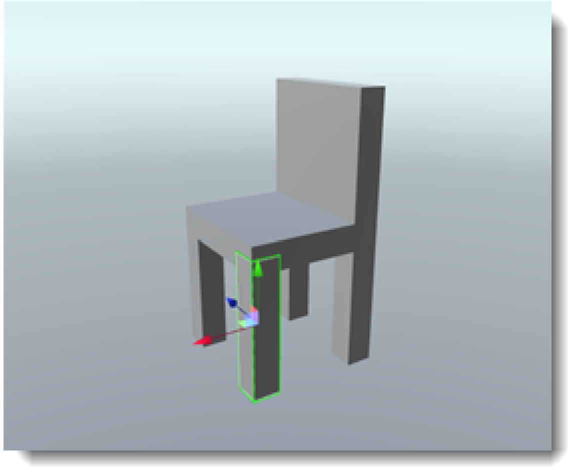
- **Mass** – the mass of the body.
- **Kinematic** – make the rigid body kinematic instead of dynamic. A dynamic body is affected by external forces such as gravity. Kinematic bodies do not fall or react when hit.
- **Velocity** – The initial linear velocity of the body.
- **Angular velocity** – The initial angular velocity of the body
- **Linear drag** – resistance of the body to linear movement, between 0 and 1.
- **Angular drag** – resistance of the body to angular movement, between 0 and 1.

If you add a collider component to the entity then the collider will be used for rigid body collision, with the center of mass being at the same location as the Rigid body entity.



Adding a single collider component and a rigid body component on the same entity is a bit limited - the collider will always be centered in the entity and you cannot move it. The solution is to put it on an entity below the rigid body component in the hierarchy. This way you can place the collider wherever you want (relative to the center of mass) and how many entities you want.

So if you'd like to make a chair consisting out of 6 colliders (4 legs + back rest + seat), you first create a root entity with a rigid body component. Then you create 6 entities with box collider components and put them as children of the root entity. Scale and position them correctly, and you're done.

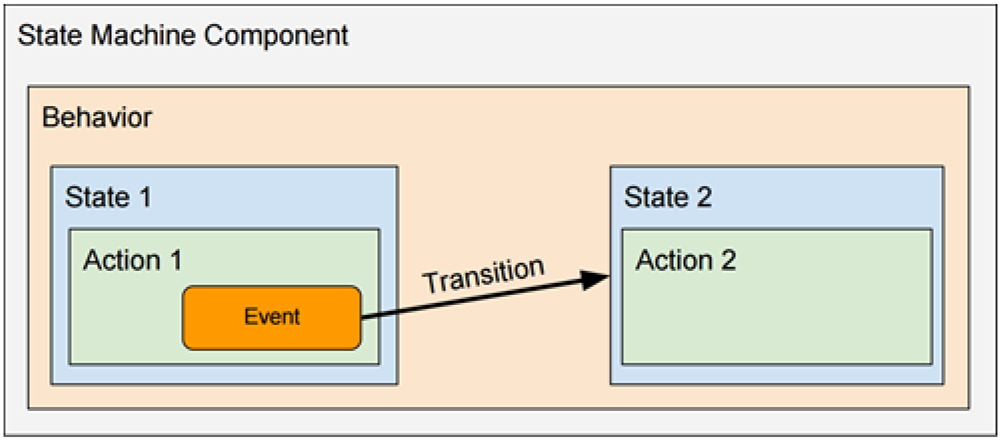


The Amazon Sumerian State Machine Component

The state machine component adds simple logic to the entity, similarly to scripts. The difference from scripts is that the State machine is much easier to use (no coding required!) and have many built-in actions.

The state machine component has two main panels, the panel for the component, and a panel for the behavior.

The state machine has a set of *behaviors*, which contain *states*, which in turn contain *actions*.



Read more about Finite State Machines on Wikipedia.

A behavior is a collection of states. A state machine component can have several behaviors, and they are independent of each other. A behavior can be seen as and behaves like an independent state machine.

Each behavior has one or more states, but only one active state. When a state is active, all of its actions will also be active. The active state can be changed by transitions in the state's actions.

A behavior always has one default state which is activated when the scene loads.

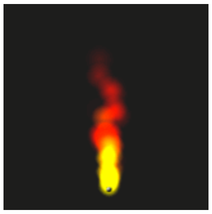
An action is some logic gets executed while its state is active. Some actions execute once and are done, while others execute once per frame. There are also actions that listen for events, and execute logic when the event happens.

Many of the actions have events, and they can trigger transitions to other states. For exaple, when the **WaitAction** executes, it sets a timer for a number of seconds. When the timer is up, it can transition to another state.

Actions

- **Add light** – adds a point light to the entity.
- **Apply force** – apply a force to the attached rigid body.
- **Apply impulse** – apply an impulse to the attached rigid body.
- **Apply torque** – apply a torque to the attached rigid body.
- **Arrow keys** – transition to other states when arrow keys are pressed.
- **Change speech volume** – change the volume of a speech component.
- **Choose/tap on entity** – select an entity.
- **Compare counter** – compare a counter with a value.
- **Compare 2 counters** – compare the value of 2 counters.
- **Camera distance** – perform a transition based on the distance to the main camera or to a location.
- **Copy joint transform** – copy a joint's transform from another entity, and apply it to this entity. This entity must be a child of an entity with an animation component.
- **Dolly zoom** – perform a dolly zoom.
- **DOM listen** – add a DOM event listener on one or many elements (specified by a query selector), and performs a transition on a given event.
- **Emit message** – emit a message (a ping) to a channel on the bus. Messages can be listened to by the Listen action, or by scripts using the SystemBus.addListener(channel, callback) function.

- **Fire FX** – make the entity emit fire. To extinguish the fire use the **Remove particles** action.

  
- **Hide** – hide an entity and its children.
- **Hover enter** – perform a transition based on whether an entity is inside a user defined box volume or not. The volume is defined by setting two points which, when connected, form a diagonal through the box volume.
- **Hover exit** – perform a transition based on whether an entity is inside a user defined box volume or not. The volume is defined by setting two points which, when connected, form a diagonal through the box volume.
- **HTML pick** – listen for a picking event and performs a transition. Can only be used on HTML entities.
- **In box** – perform a transition based on whether an entity is inside a user defined box volume or not. The volume is defined by setting two points which, when connected, form a diagonal through the box volume.
- **Increment counter** – increment a counter with a value.
- **In view** – perform a transition based on whether the entity is in a camera's frustum or not.
- **Key down** – listen for a key press and performs a transition.
- **Key pressed** – listen for a key press event and performs a transition. Works over transition boundaries.
- **Key up** – listen for a key release and performs a transition.
- **Log message** – print a message in the debug console of your browser.
- **Look at** – reorient an entity so that it's facing a specific point.
- **Mouse down** – listen for a mouse button press and perform a transition.
- **Mouse move** – listen for mouse movement and performs a transition.
- **Mouse pressed** – listen for a mouse button press event and performs a transition. Works over transition boundaries.
- **Mouse up** – listen for a mouse button release and performs a transition.
- **Move** – move the entity.
- **Mute** – mute all sounds globally.
- **Next frame** – transition to a selected state on the next frame.
- **Pause animation** – pause skeleton animations.
- **Pause particle system** – pause particle system.
- **Pause sound** – pause a sound.
- **Pause timeline** – pause a timeline.
- **Pick** – listen for a picking event on the entity and perform a transition.
- **Pick and exit** – listen for a picking event on the entity and open a new browser window.
- **Play sound** – listen for a picking event on the entity and open a new browser window.
- **Random transition** – perform a random transition.
- **Remove** – remove the entity from the world.
- **Remove light** – remove the light attached to the entity.
- **Remove particles** – remove any particle emitter attached to the entity
- **Resume animation** – continue playing a skeleton animation.

- **Rotate** – rotate the entity with the set angles (in degrees).
- **Scale** – scales the entity.
- **Set animation** – transition to a selected animation.
- **Set animation offset** – change the animation clip offset.
- **Set background color** – change the background color.
- **Set counter** – change a counter to a value.
- **Set HTML text** – change the contents of an HTML element.
- **Set light properties** – change various properties of a light.
- **Set light range** – change the range of a light.
- **Set material color** – change the color of a material.
- **Set render target** – renders what a camera sees on the current entity's texture.
- **Set rigid body angular velocity** – change the angular velocity of a rigid body.
- **Set rigid body position** – change the position of a rigid body.
- **Set rigid body rotation** – change the rotation of a rigid body.
- **Set rigid body velocity** – change the velocity of a rigid body.
- **Set rotation** – change the rotation of an entity.
- **Set timelime time** – jump to a point on a timeline.
- **Set animation time scale** – change the time scale for the current animation.
- **Shake** – shake the entity and optionally perform a transition.
- **Show** – make an entity visible.
- **Smoke FX** – make an entity emit smoke. To cancel the smoke emitter use the **Remove particles** action.

- **Sound fade in** – fade in a sound.
- **Sound fade out** – fade out a sound and stop it.
- **Sprite animation** – start a sprite animation.
- **Start particle system** – start a particle emitter.
- **Start speech** – start a speech component.
- **Start timeline** – start a timeline.
- **Stop particle system** – stop a particle emitter.
- **Stop sound** – stop a sound.
- **Stop speech** – stop a speech component.
- **Stop timeline** – stop a timeline.
- **Switch camera** – switch to a different camera.
- **Toggle full screen** – toggle fullscreen on/off. Note that in most browsers this must be initiated by a user gesture. For example, click or touch.
- **Toggle mute** – toggles mute of all sounds globally.
- **Toggle post FX** – enable or disable post fx globally.
- **Transition** – transition to a selected state.
- **Listen** – perform a transition on receiving a system bus message (a ping) on a specific channel.

- **Trigger enter** – transition when a trigger volume is entered.
- **Trigger leave** – transition when a collider is leaving the trigger volume.
- **Tween light** – tween the color of the light.
- **Tween look at** – transition the entity's rotation to face the set position.
- **Tween material color** – tween the color of a material.
- **Tween move** – transition to the set location.
- **Tween material opacity** – tween the opacity of a material.
- **Tween rotate** – transition to the set rotation, in angles.
- **Tween scale** – transition to the set scale.
- **Tween texture offset** – smoothly change the texture offset of the entity.
- **Unmute** – unmute all sounds globally.
- **Wait** – perform a transition after a specified amount of time. A random time can be set, this will add between 0 and the set random time to the specified wait time.
- **WASD keys** – transition to other states when the WASD keys are pressed.

The Amazon Sumerian Script Component

You can add scripts to any entity. A script component can contain multiple scripts. Scripts run in order from top to bottom and you can adjust the order in the script component properties.

To support re-use, you add an instance of a script to the script component, not the script itself. The instance contains the state and parameters of the script, letting you add multiple instances of the same script with different behavior on each, based on the arguments provided.

Properties

- **Enabled** – clear the checkbox to disable a script.
- **Instance of** – each script instance in the list has a reference to the script it is using. Choosing the script will take you to the scripts' own panel.
- **Parameters** – any parameters defined in the script's `parameters` array (p. 61) appear here. Adjust the values to customize the behavior of this script instance.

To structure your parameters, you can store them in JSON file. Start by defining the parameters in the script itself. Then add the JSON file and reference it from the script settings.

Example Script with JSON parameter

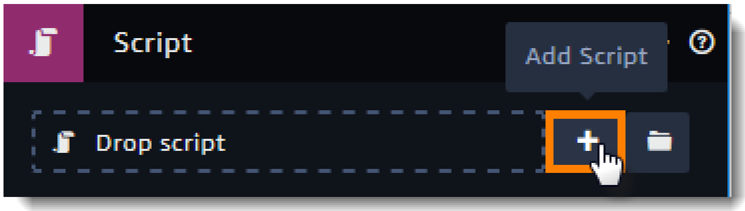
```
varsetup=function(args,ctx){
    console.log(args.myJsonParameter);// Prints the parsed JSON data
};

var parameters=[{
    key:'myJsonParameter',
    type:'json'
}];
```

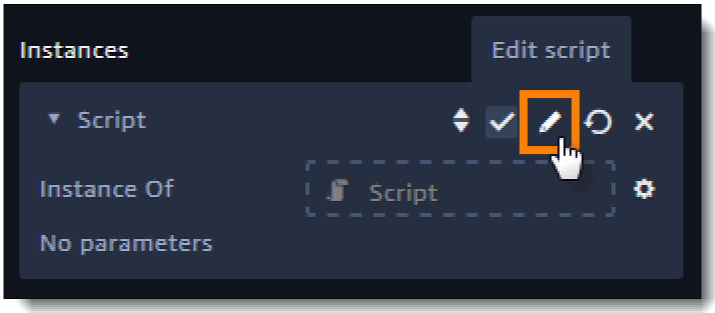
To create a script with JSON parameters

1. Create a blank entity (p. 32).
2. Choose **Add component** and then choose **Script**.

3. Choose **Add script** and then choose **Custom**.



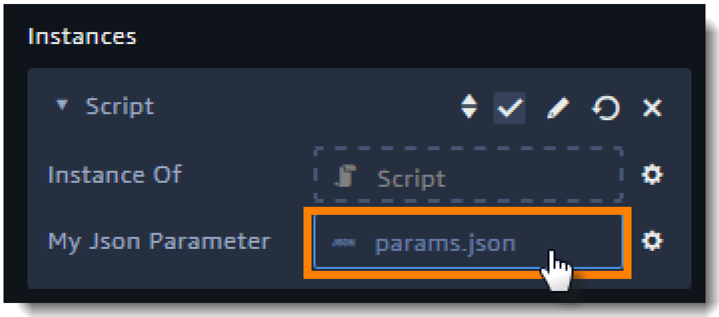
4. Choose **Edit script**.



5. Replace the default `parameters` declaration with the following.

```
var parameters=[{
    key:'myJsonParameter',
    type:'json'
}];
```

6. Return to the script settings. The settings automatically update to include the JSON parameter.
7. Drop a JSON file in the parameter field.



The Amazon Sumerian Timeline Component

The timeline component animates properties of an entity over time, such as the transform.

When you add the timeline component to an entity, you'll get a component panel with just one button. Choose the *Toggle timeline* button to open the timeline.

To add an entity to your timeline, drag it from the Hierarchy view to the drop area at the left bottom in the timeline. The entity will appear to the left in the timeline. View the entity properties available for animation by clicking on it. Each property is called a *Channel*.

Note that you can *not* add the owner entity to the timeline.

To change a property value over time, you add *Keyframes* to a *Channel*. A Keyframe has a position on the timeline, a value and an easing function. As time go by, the property value will be set to the value given by the keyframes, interpolated by the easing functions.

The timeline has a start and an end. You can set the duration of the timeline and toggle looping in the right bottom corner.

Event channels are used to emit SystemBus events at given points in time. Choose *Add Event Channel* to add a channel for an event. You will be prompted for an event name to use.

To fetch these events, you can use `SystemBus.addListener` in a Script or use the *Listen Action* in the State Machine. Make sure to listen for the same event name that you specified when adding the channel.

Assets

The assets panel collects shareable assets in the scene. When you add an asset to a scene, Sumerian automatically adds it to a *default pack* in the assets panel. Entities that you create are not automatically added, but you can drag them from the entities panel into the assets panel to create an entity asset.

Asset Types

- **Entity**
- **Mesh**
- **Skeleton**
- **Entity**
- **Sound**
- **Script**
- **JSON**
- **Skybox**
- **Behavior**
- **Texture**

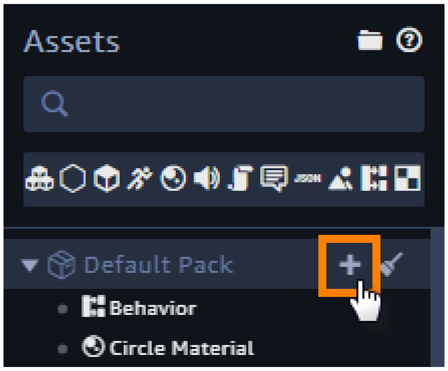
To add an asset to a scene, you can drag it from your desktop directly onto the canvas. Depending on the file type, the editor creates an entity in the entities panel, and one or more assets. For example, when you add a JPG image to a scene, you get an image entity in the scene, a material asset, and a texture asset.

To add an asset to a scene

1. Open a scene in the Sumerian editor.
2. Drag a file from your desktop file browser onto the canvas.

   or

   Create a blank asset by clicking the plus icon next to the default pack. Select the pack name to see the icon.



3. Choose the asset in the assets panel and modify it by using the options in the inspector panel.

Packs

You can organize your assets and share them between scenes by creating a *pack*. Create a pack in the assets panel and move or copy assets into it.

Packs support the following actions.

- ➕ – create a new asset in the pack.
- △ – export the pack to the asset library.
- 🧹 – delete any assets in the pack that are not used in the scene.
- 🗑 – delete the asset pack.

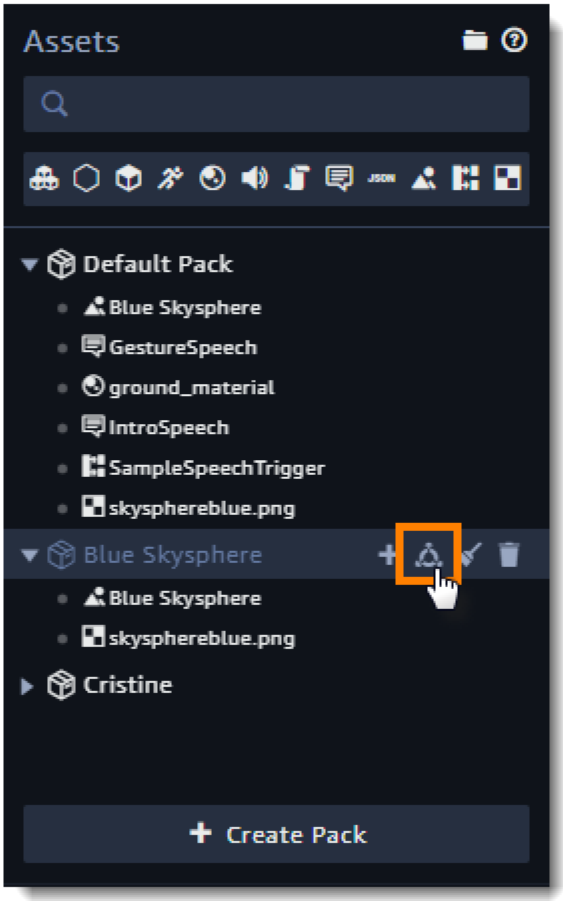
Exporting a pack adds it to the project that you choose. If you do not have a project yet, create one in the dashboard (p. 7). You can then use the dashboard to copy or move the pack (p. 9) to a different scene or project. Exported packs are not tied to the in-scene pack or its assets.

To add an asset to a pack and export it

1. Open a scene in the Sumerian editor.
2. Under **Assets**, choose **Create pack**.
3. Choose the pack and modify the name, description, tags, and custom attributes in the inspector panel.
4. Drag an asset from the default pack into the custom pack.

   or

   Duplicate the item by selecting it and then clicking the duplicate icon. Drag the duplicate into the custom pack.
5. Select the custom pack and then click the asset library icon △, or choose **Add to asset library** in the inspector panel.



6. Choose an asset type (p. 52) for the asset pack.
7. Choose **Add to asset library**.
8. Choose a project and then choose **Select**.

Models

When you import a model, Sumerian converts it into a format compatible with the Sumerian engine. The process can take some time depending on the model size and format. You can follow the progress of the conversion in the status bar.

Triangulation
Meshes will automatically be triangulated during the conversion. Triangle meshes are a requirement for the engine.

To optimize the model importing process, remove unneeded data by deleting the object history and freezing transformations in your modeling tool. Avoid using geometric transformations if possible. If your model has animations, bake the animations into the model and avoid using constraints.

You can import models in the following formats.

File Formats

- **FilmBox** – `.fbx`

www.autodesk.com/products/fbx/overview

- **Wavefront OBJ** – `.obj`

  en.wikipedia.org/wiki/Wavefront_.obj_file

The Sumerian engine supports the following model features.

Model Features

- **Vertex colors** – Per-vertex colors or per-face-vertex colors is supported.

  When the mesh data contains vertex colors, a slider will be available on the mesh's material panel under the diffuse channel. Here you are able to blend between the set diffuse map or color and the vertex color.
- **UV maps** – If two are available, the second one can be used for e.g. light maps or ambient occlusion maps.

  In the editor, you are able to apply these textures on the ambient channel in the material panel.
- **Tangents** – If no tangent data is provided, this will be generated during the conversion.
- **Normals** – If no normal data is provided, interpolated normals will be generated during the conversion.
- **Skeleton Animations** – Animation via skeleton mesh deformation is supported. You can provide several animations in one file.

  Shader limitations

  - The maximum number of weights per vertex is 4. If more are provided, the ones with the least values are removed.
  - Keeping the joint count low will allow supporting a broader set of hardware.

  If you already have converted a model with skeleton animations into the editor, and afterwards added more animations in your modeling tool, you can to add those new animations onto the existing model in the editor.

  This is done by dropping the file upon the animation panel's animation state drop area. This issues the file upload as usual, but during conversion, only the animation data is exported.

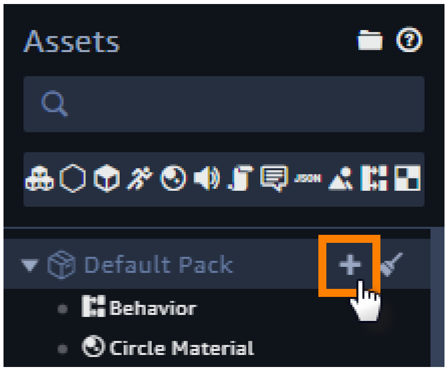
  Note that the underlying skeleton rig must be the same for this to work. If you have done changes to the rig you will need to re-import the model through the regular process.
- **Embedded textures (FBX)** – When exporting to the FBX binary format, you can embed textures into the resulting file.

Skybox

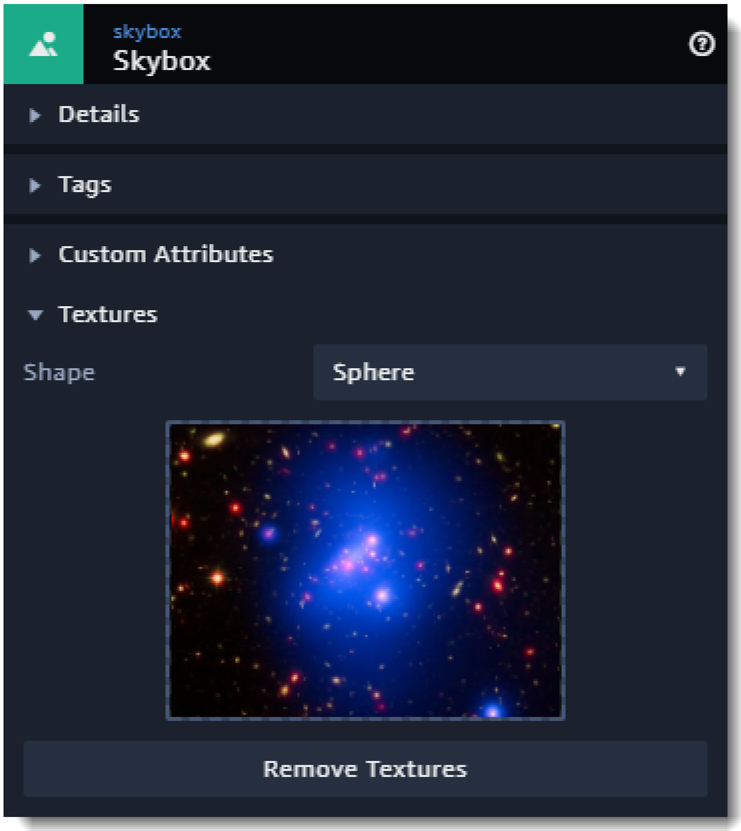
A skybox is a texture that applies to the background of a scene, to show the sky, space or an enclosing structure. A skybox can be a single texture that wraps onto a sphere, or six textures that wrap onto a cube. Add a skybox to your scene in the scene's environment settings (p. 26).

To create a skybox

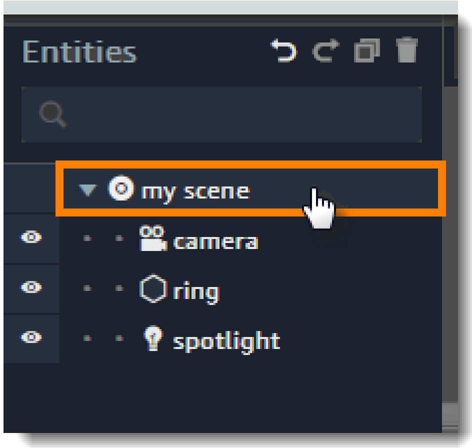
1. Open a scene in the Sumerian editor.
2. Clicking the plus icon next to the default pack. Select the pack name to see the icon.



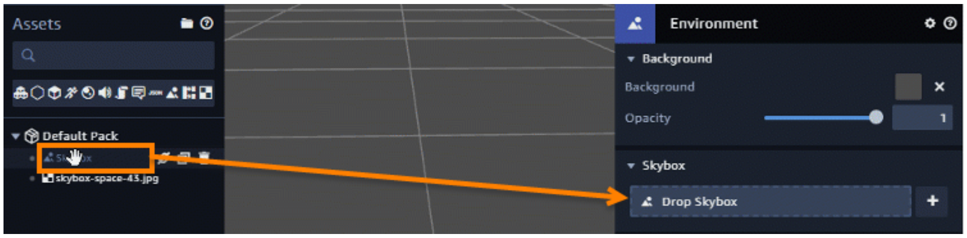
3. Choose **Skybox**.
4. Choose the shape of the skybox.
5. Drop a texture asset or image file on each section of the skybox.



6. Choose the root node in the **Entities** panel.



7. Choose **Environment**.
8. Drop the skybox asset from the assets panel onto the **Skybox** field.



Media

You can import media files to use as textures or audio objects. Files up to 10MB are supported in the following formats.

Images

- CRN
- DDS
- JPG, JPEG
- PNG
- SVG
- TGA

Sound

- OGG
- MP3
- WAVE, WAV

Video

- MP4
- OGV

- WEBM

Text

- JS
- JSON

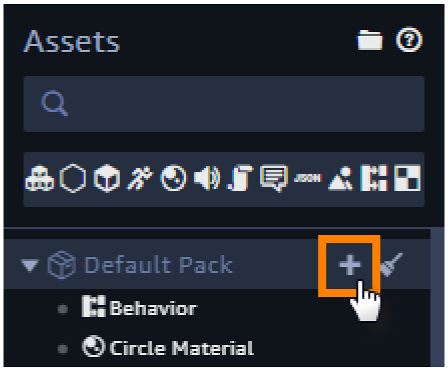
Scripting

Add scripts to your scene to update your scene based on user input or events. You can use scripts to access the DOM, create and modify entities with the Sumerian engine library, or use the AWS SDK for JavaScript to access AWS services and resources.

> **Note**
> Reference documentation for the Sumerian engine library is available on the Sumerian website.

To create a blank script

1. Open a scene in the Sumerian editor.
2. Create a blank asset by clicking the plus icon next to the default pack and then click **Script**. Select the pack name to see the icon.



3. Press **j** to open the text editor.
4. Choose the new script under documents. Use the pencil icon next to the script name to change its name.

The script template includes 7 methods and a parameters (p. 61) array. The methods correspond to a scene's lifecycle events and are called by the engine at the following times.

- `setup` – when scene playback starts.
- `fixedUpdate` – on every physics update.
- `update` – on every render frame.
- `lateUpdate` – after calling all `update` methods in the scene.
- `enter` – on a state machine script action, when the state is entered.
- `exit` – on a state machine script action, when the state is exited.
- `cleanup` – when scene playback stops.

Topics

Built-in Scripts

The editor also has several built-in scripts that provide standard functionality like camera, keyboard, and mouse controls.

Camera scripts

- **Orbit camera control** – lets the user orbit the scene by holding a mouse button and moving the mouse.
- **Orbit and pan control** – lets the user orbit the scene with one mouse button and pan the camera with another.
- **Fly control** – lets the user zoom and pan with the keyboard.
- **Axis-aligned camera control** – Move the camera to a fixed distance away on the X or Z axis.
- **Pan camera control** – lets the user pan the camera by holding a mouse button and moving the mouse.
- **Mouse look control** – lets the user look around by holding a mouse button and moving the mouse.
- **WASD control** – lets the user walk around on the XZ plane with the keyboard.

Object scripts

- **Button** – lets the user click on an object to open a URL.
- **Pick and rotate** – lets the user grab an object and manipulate its orientation.
- **Lens flare** – generates a lens flare when the user looks at an object.

The Context Object

You can use the context object, `ctx` to store your script data during the script life time. The context is created upon setup() and cleared on cleanup() and is passed into all of the script functions. It has a few pre-defined properties:

Properties

- `entity` (`Entity`) – the entity to which the script is attached.
- `entityData` (`Object`) – a data object shared between all scripts on the entity.
- `activeCameraEntity` (`Entity`) – the currently active camera entity.
- `domElement` (`HTMLCanvasElement`) – the WebGL canvas element.
- `playTime` (`number`) – the elapsed time since scene start.
- `viewportHeight` (`number`) – the height of the canvas.
- `viewportWidth` (`number`) – the width of the canvas.
- `world` (`World`) – the world object.
- `worldData` (`Object`) – a data object shared between all scripts in the world.

Some of the properties on `ctx` are shared between scripts. *entityData* is shared by all scripts on the entity and *worldData* is shared by all scripts. They are all initially empty, and can be used to store any kind of data

For example, if we'd like to define a property called *acceleration*, we could make it available on three levels:

```
// Only accessible to the script that defined the property
```

```
ctx.acceleration=9.82;

// Accessible to all scripts on the entity
ctx.entityData.acceleration=9.82;

// Accessible to all scripts
ctx.worldData.acceleration=9.82;
```

The built-in context properties also contain some convenience functions. For example, the `world` object lets you search for entities based on their tags. You can get all entities with a specific tag with `ctx.world.by.tag`:

```
var entities = ctx.world.by.tag('myTag');
```

Parameters and Arguments

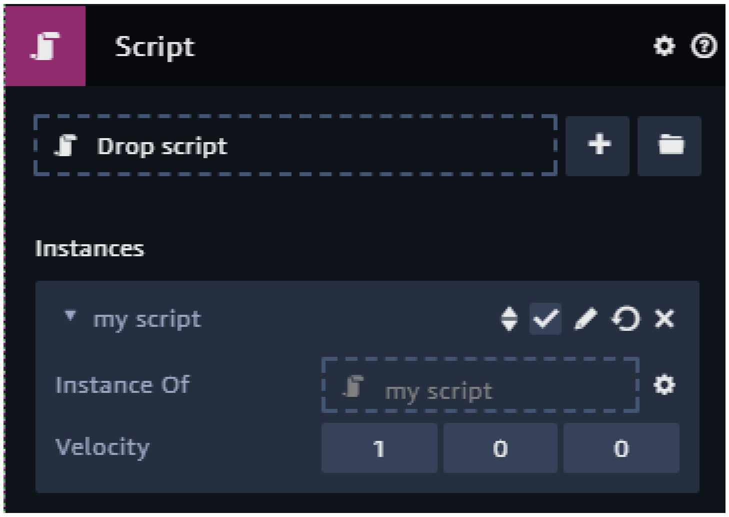
Parameters let you create scripts that are customizable by adding fields to the script properties in the editor. For example, the following script defines a parameter named `Velocity` that takes 3 numbers (a `vec3` parameter (p. 62)).

```
var setup = function(args, ctx){
    console.log(args.velocity);
};

var parameters = [
  {
    name : "Velocity",
    key : "velocity",
    type : "vec3",
    default : [1,0,0]
  }
];
```

During the setup phase, the script reads the parameter values from the `args` object and prints them to the console.

When you add an instance of the above script to an entity, the editor shows a **Velocity** field that accepts three values and reflects the default value.



Instance Of: my script

Velocity: 1 0 0

Parameter Format

Parameters are objects with the following required and optional fields.

Required fields

- **key** [string] – a unique key used to store and retrieve the parameter values in the `args` object.
- **type** [string] – the parameter type (p. 62).
- **default** – the default value or values for the parameter.

Optional fields

- `name` [string] – the label for the parameter field shown on instances of the script. If you don't specify a name, the `key` is used to generate the label.
- `control` [string enum] – the control type.
  - `slider` – a slider control.
  - `color` – a color wheel.
  - `select` – a drop down listing the values in the `options` field.
  - `jointSelector` – a drop down listing the joints on the animation component on the script's parent entity.
- `description` [string] – the description shown when you hover over the parameter.
- `options` [array] – an array of possible values for a `select` control.
- `min` and `max` [number] – the minimum and maximum values for an `int` or `float` parameter.
- `decimal` [number] – the number of significant digits for a `float` parameter.
- `step` [number] – the incremental value that `float` values snap to.
- `precision` [number] – the number of significant digits for `float` values.
- `exponential` [boolean] – set to `true` to distribute the values on a `slider` control logarithmically.

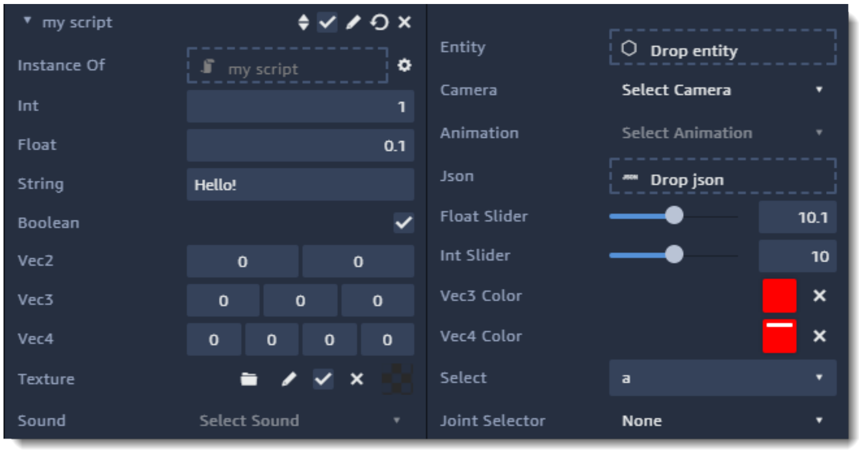
Parameter Types

The type property must be set to one of a few predefined strings, each corresponding to a type of parameter.

- `int` – Integer number variable (e.g. `1`).
- `float` – Number variable (e.g. `3.14`).
- `string` – String (e.g. “`HelloGoo`”).
- `boolean` – boolean (`true` or `false`).
- `vec2`, `vec3`, `vec4` – an array of 2, 3, or 4 numbers.
- `texture`, `sound`, `entity`, `camera`, `animation`, `json` – an asset of the specified type (p. 52).

All types in action, including a sample script:

```
var parameters = [
    {type: 'int', key: 'int', 'default': 1, description: 'Integer input'},
    {type: 'float', key: 'float', 'default': 0.1, description: 'Float input'},
    {type: 'string', key: 'string', 'default': 'Hello!', description: 'String input'},
    {type: 'boolean', key: 'boolean', 'default': true, description: 'Checkbox'},
    {type: 'vec2', key: 'vec2', 'default': [0, 0], description: 'Vector2 input'},
    {type: 'vec3', key: 'vec3', 'default': [0, 0, 0], description: 'Vector3 input'},
    {type: 'vec4', key: 'vec4', 'default': [0, 0, 0, 0], description: 'Vector4 input'},
    {type: 'texture', key: 'texture', description: 'Texture asset drop area'},
    {type: 'sound', key: 'sound', description: 'Sound asset drop area'},
    {type: 'entity', key: 'entity', description: 'Entity drop area'},
    {type: 'camera', key: 'camera', description: 'Camera drop down'},
    {type: 'animation', key: 'animation', description: 'Animation state from the Animation
 component on the same entity'},
    {type: 'json', key: 'json', description: 'JSON asset drop area'},
    {type: 'float', control: 'slider', key: 'floatSlider', 'default': 10.1, min: 5, max:
 15, exponential: false, decimal: 1, description: 'Float slider input'},
    {type: 'int', control: 'slider', key: 'intSlider', 'default': 10, min: 5, max: 15,
 exponential: false, description: 'Integer slider input'},
    {type: 'vec3', control: 'color', key: 'vec3Color', 'default': [1, 0, 0], description:
 'RGB color input'},
    {type: 'vec4', control: 'color', key: 'vec4Color', 'default': [1, 0, 0, 1],
 description: 'RGBA color input'},
    {type: 'string', control: 'select', key: 'select', 'default': 'a', options: ['a', 'b',
 'c'], description: 'Dropdown/select'},
    {type: 'int', control: 'jointSelector', key: 'jointSelector', description:  'Joint
 select from the animation component on a parent entity'}
];
```



my script

Instance Of	my script	Entity	Drop entity
Int	1	Camera	Select Camera
Float	0.1	Animation	Select Animation
String	Hello!	Json	Drop json
Boolean	✓	Float Slider	10.1
Vec2	0 0	Int Slider	10
Vec3	0 0 0	Vec3 Color	🟥
Vec4	0 0 0 0	Vec4 Color	🟥
Texture		Select	a
Sound	Select Sound	Joint Selector	None

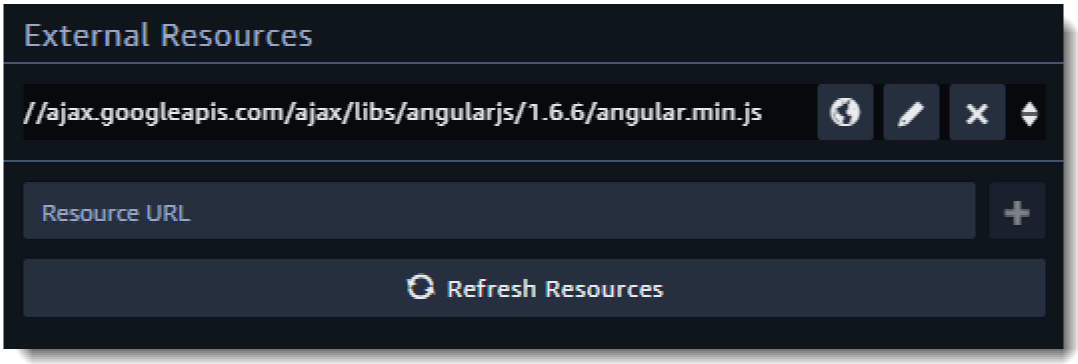
External Dependencies

You can load external JavaScript libraries from the web into your script, enter a URL in the left panel of the script editor, and click the + button.

Note that the URLs you enter should start with *//* and not *http://* or *https://*.

To declare external dependencies

1. Open a scene in the Sumerian editor.
2. Press J to open the text editor.
3. Choose a script asset in the **Documents** list.
4. Under **External resources**, enter a URL starting with // (excluding the protocol).
5. Click the plus icon to add the library to the list.


External Resources
//ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js
Resource URL
Refresh Resources

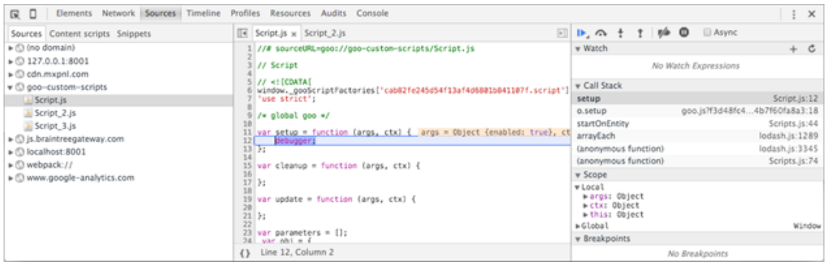
The JavaScript dependencies will be loaded and executed immediately inside the editor (click “Refresh resources” to re-download and execute). In your published scene, all dependencies will be loaded and executed during the loading phase.

Debugging

To debug a script in the Sumerian editor, use the built in tools in your browser. In Google Chrome, open Developer Tools by pressing ALT-CMD-J on Mac or F12 on Windows.

Open the *Sources* panel at the top of Devtools. To the left you can see all scripts loaded in the browser. If you have a script in your scene, it will be listed in below *sumerian-custom-scripts*. Choose your script to view it.

The simplest way to start debugging a script is by adding a `debugger;` statement in your Custom Script in the editor. If you have Devtools open, and this statement is executed, Devtools will automatically go to the file and line where your statement is.



```
//# sourceURL=goo://goo-custom-scripts/Script.js

// Script

// <![CDATA[
window._gooScriptFactories['cab82fe245d54f13af4d6801b841107f.script']
'use strict';

/* global goo */

var setup = function (args, ctx) {  args = Object {enabled: true}, ct
	debugger;
};

var cleanup = function (args, ctx) {

};

var update = function (args, ctx) {

};

var parameters = [];
 var obj = {
```

```
setup                          Script.js:12
o.setup                        goo.js?f3d48fc4...4b7f60fa8a3:18
startOnEntity                  Scripts.js:44
arrayEach                      lodash.js:1289
(anonymous function)           lodash.js:3345
(anonymous function)           Scripts.js:74
```

```
▼Local
  ▶args: Object
  ▶ctx: Object
  ▶this: Object
▶Global                        Window
```