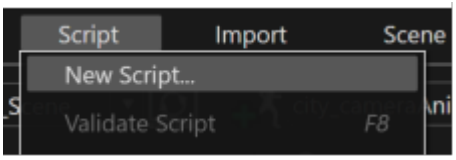
Scripting

This section provides an introduction to SceneComposer's scripting user interface.

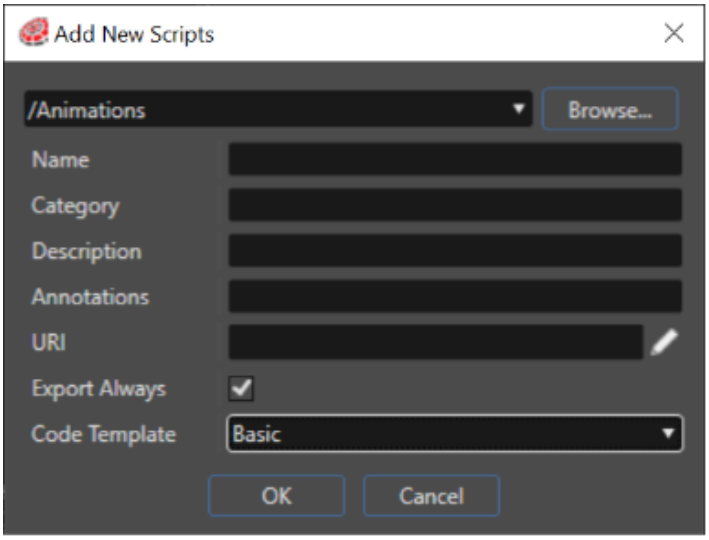
- New Script
- Script Editor and Script Validation
- Script Components
- Script Execution
- Support for Object Reference Properties

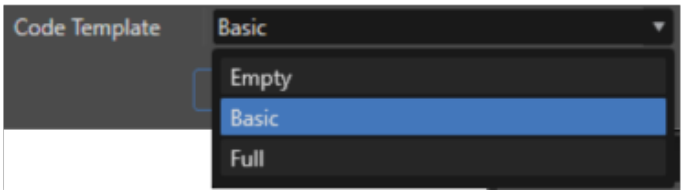
New Script

A new script is created using the "New Script" menu item.



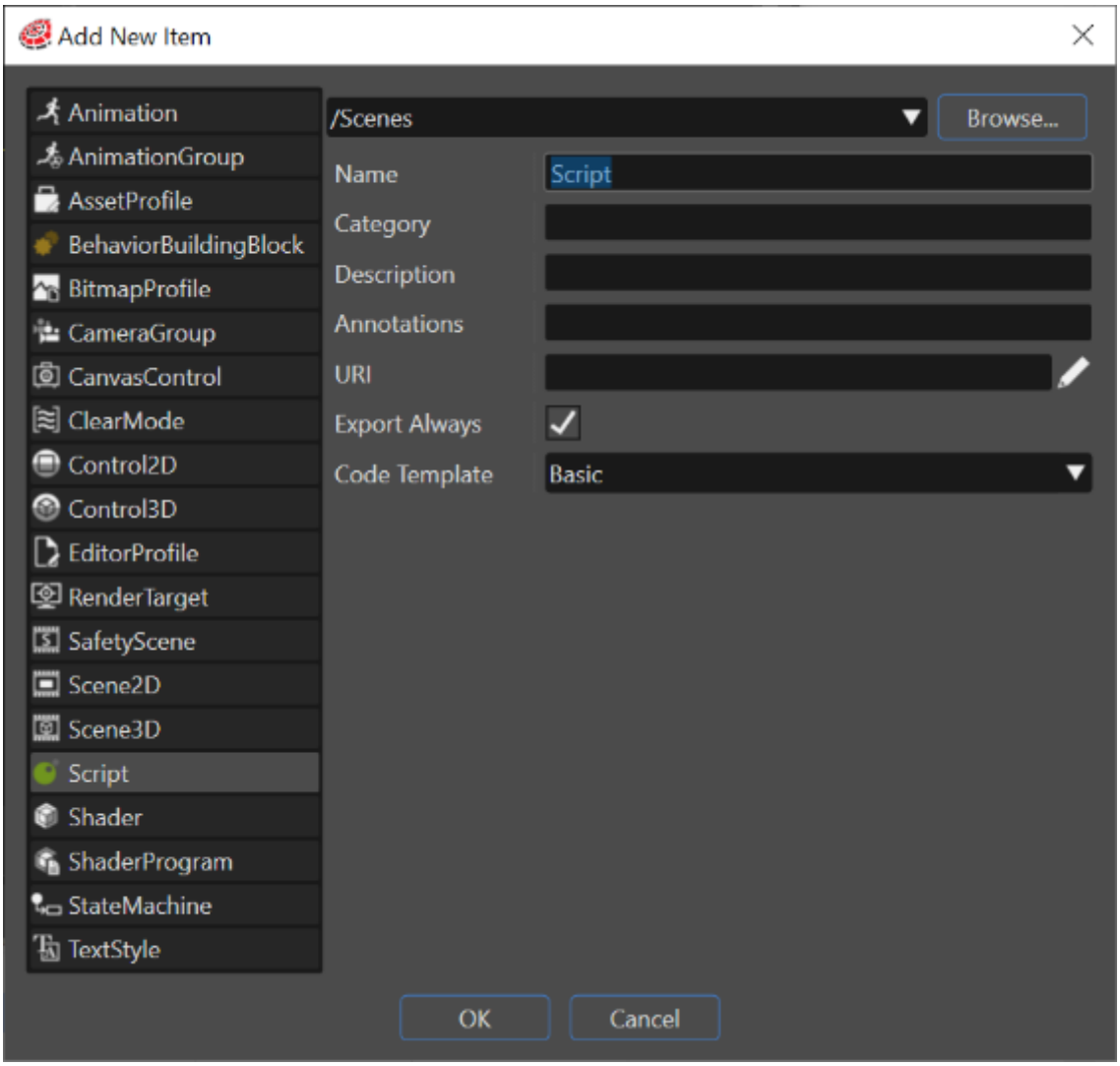
Selecting the item will prompt a dialog to enter a name.



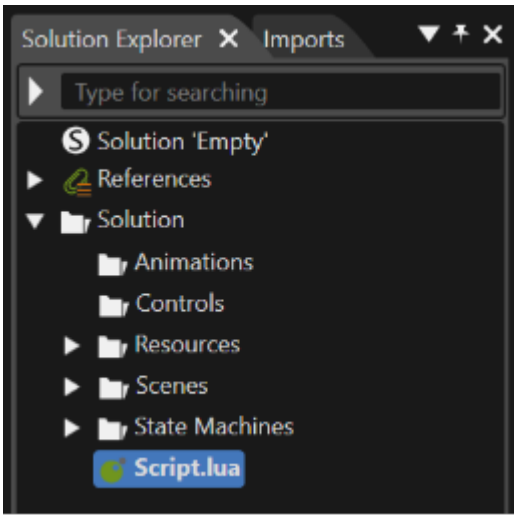


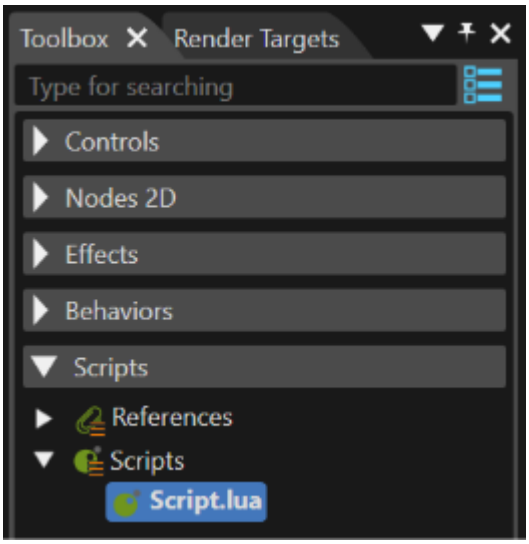
Depending on the option 'Code Template' in the dialog, the newly created script will either be empty (Code Template: Empty), contain commonly used callback functions (Code Template: Basic), or contain the full set of callback functions for the user to implement (Code Template: Full).

Alternatively, scripts can be added to a solution via "Add" > "New item" option by using the Solution Explorer context menu.



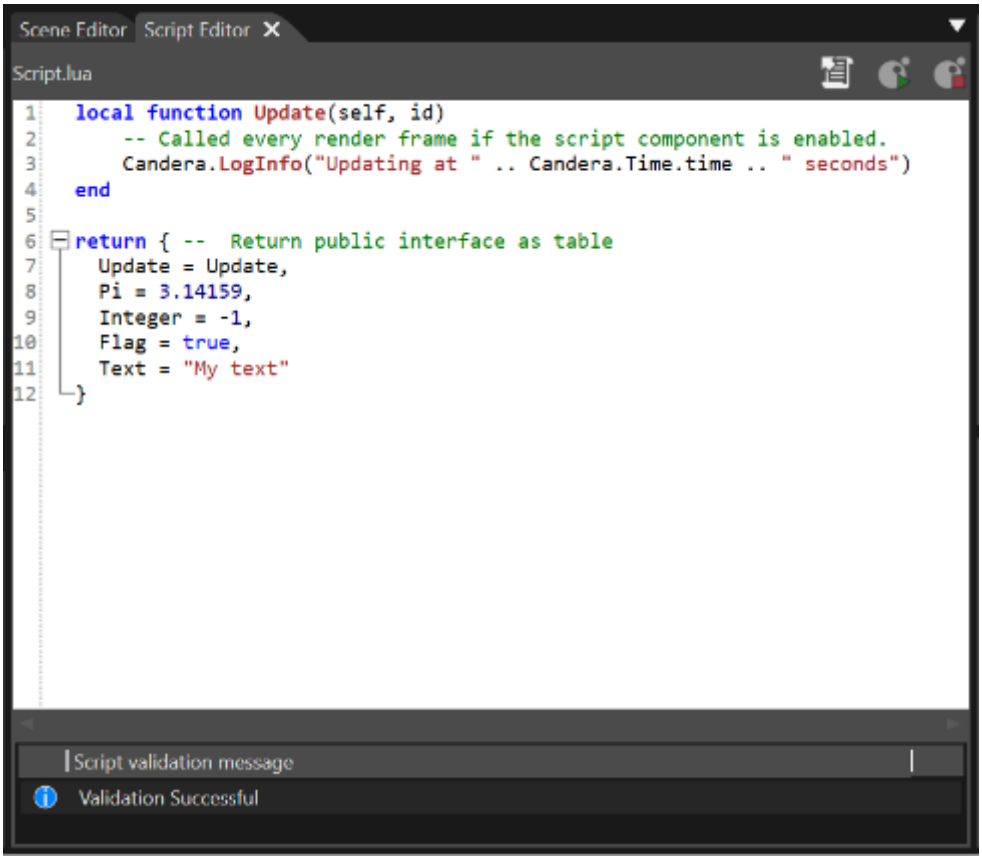
Scripts will show up in the Solution Explorer and the Scripts section of the Toolbox with the postfix ".lua".



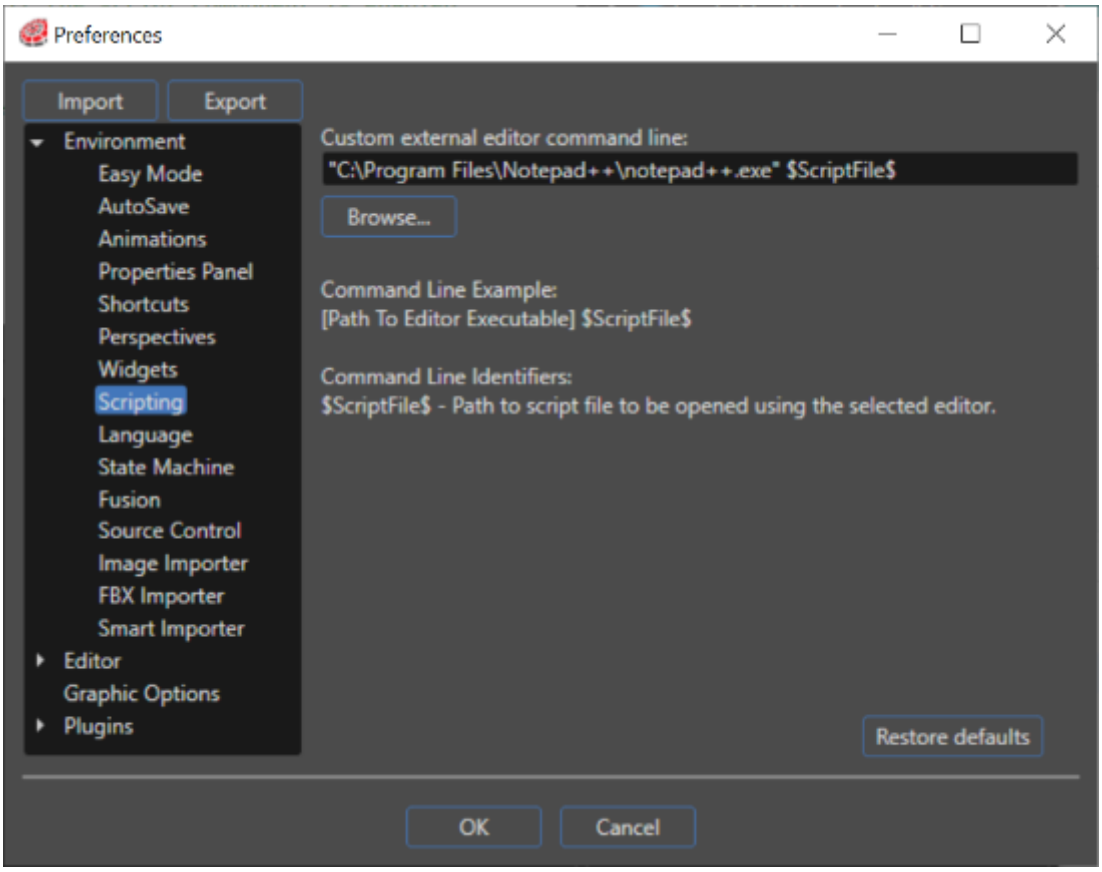

Toolbox
Render Targets
Type for searching
Controls
Nodes 2D
Effects
Behaviors
Scripts
References
Scripts
Script.lua

Script Editor and Script Validation

Double clicking on a script in the Solution Explorer, Toolbox, or on a script component, opens the Script Editor. The editor supports syntax highlighting for Lua.



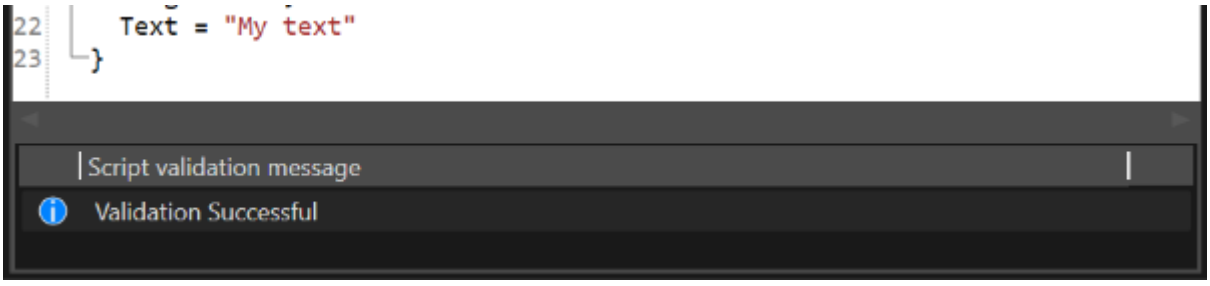
The user can edit scripts by using an external editor, too. The script code will be automatically updated and validated in SceneComposer after any modification in the external editor. Any error will be shown. To set up the path to the external editor a specific section is available in "Preferences" panel (see the image bellow).



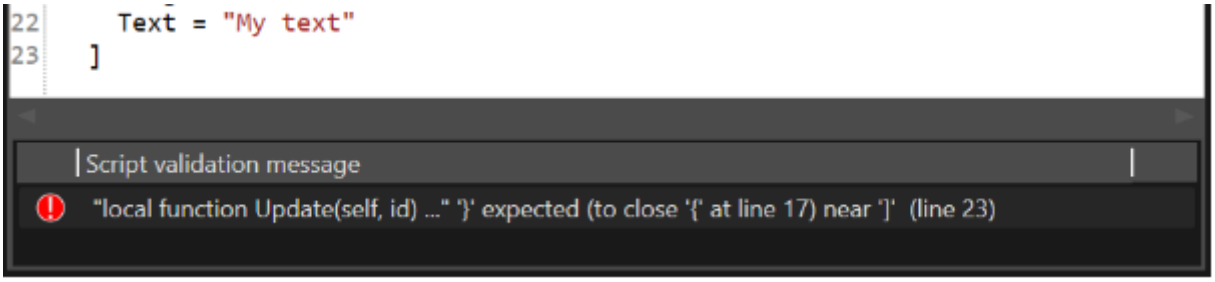
Once a script has been input (a tutorial covering script development can be found here), it needs to be validated. The operation of starting scripts will validate any script that has not been validated yet. Validation of a script means that the script is passed to Lua's virtual machine which checks if the script is syntactically correct.

After a successful validation no errors will be shown.

> If a script has a component which cannot be validated, the script system will not start



After an unsuccessful validation the Lua error message is displayed with the offending line. Clicking on the error message will position the cursor in that line in the editor.



When a scene is loaded, all its scripts are validated automatically. Any scripts that were changed or added after loading the scene, have to be validated manually.

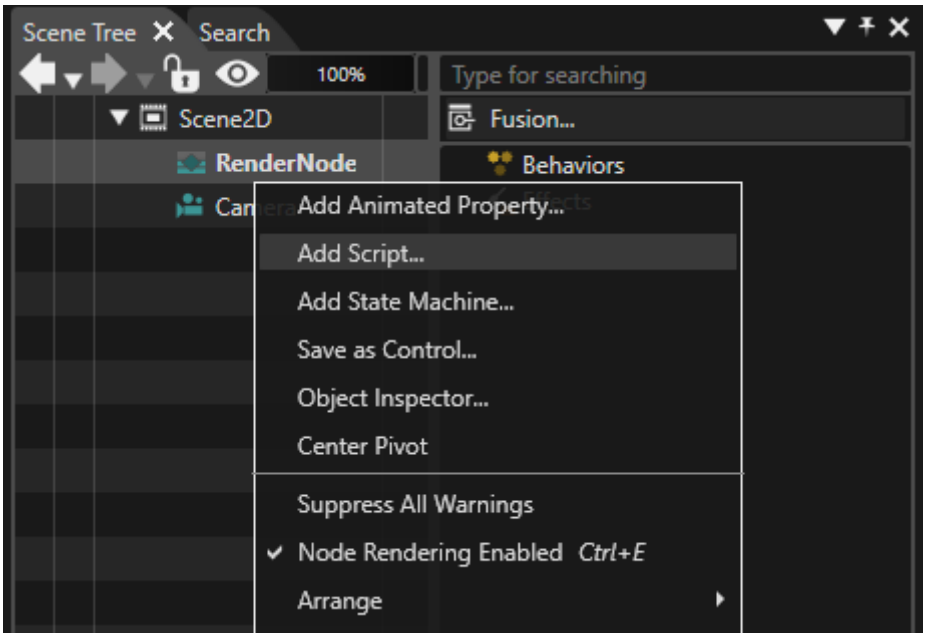
> The "Script Editor" is not cleared if the script loaded inside is deleted from solution.

Script Components

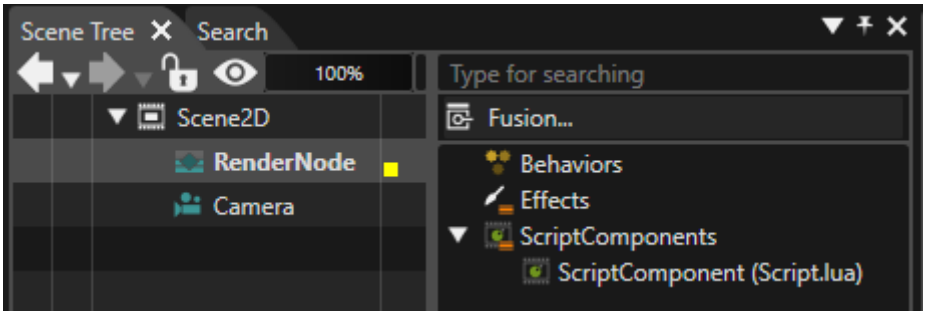
Scripts themselves are assets just like textures or shaders. This means they have to be referenced in the scene so that they can be executed and operate on scene objects. This is were script components come in. A script component is an instance of a script that can be attached to nodes using customizable parameters.

New script component

Attaching a script to a node automatically creates a script component that instances the script. Attaching scripts can be done by dragging and dropping the script directly on the node from the Solution Explorer or Toolbox. It can also be attached by selecting the node using a right mouse button press and using "Add Scripts... " from its context menu.



After attaching a script to a node, it can be found on the right side of the Scene Tree tab when selecting its corresponding node. Multiple scripts (including the same script) can be attached to the same node.



To delete a script component, select it and press the 'Del' key. Alternatively, select it by pressing the right mouse button and select 'Delete' from the context menu.

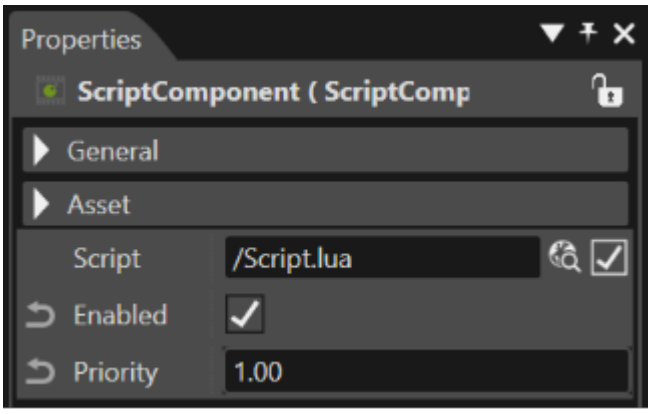
> Locking a Script item will not also lock the Script Editor.

Script Component Properties

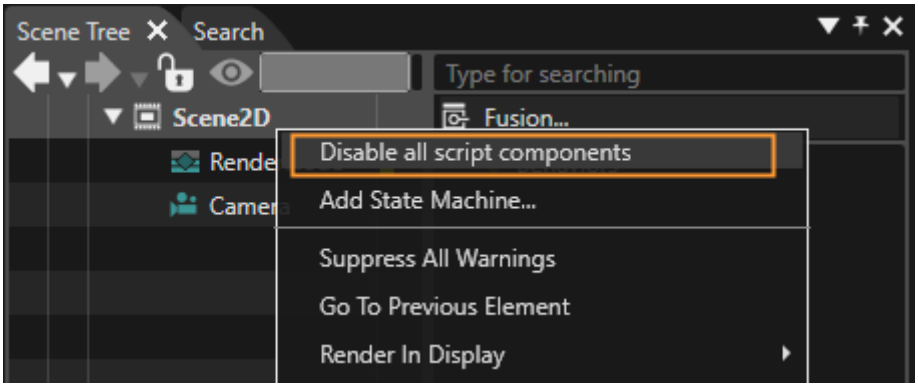
Every script component has the following 2 properties:

- **Enabled** : If this flag is set, the script component will receive callbacks, otherwise not. This flag can be set by the user or by scripts using Candera.SetEnabled.
- **Priority** : This value determines in which order script components receive callbacks. Higher values mean higher priority. Highest priority script components are executed first. The priority is a floating point value, so it's easy to place a new component inbetween other components prioritywise without having to reassign multiple values. Negative values are allowed as well.

> The "Enabled" flag and the "Priority" can also be changed in realtime affecting the execution of scripts. After the script system is stopped the properties "Enabled" and "Priority" will display their initial values.

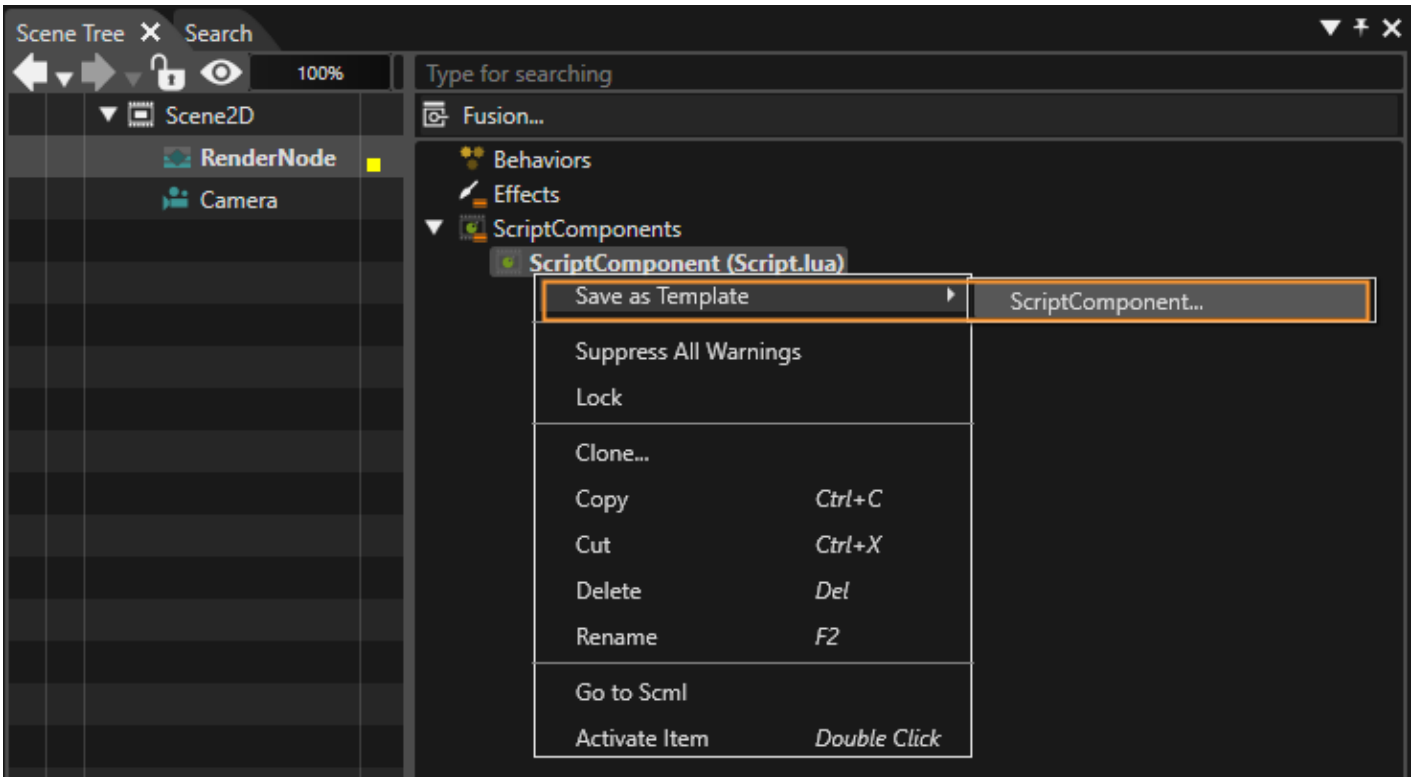


Another possibility to enable/disable script components is by using the option "Enable/Disable all script components" which is available in the context menu of any scene with script components.

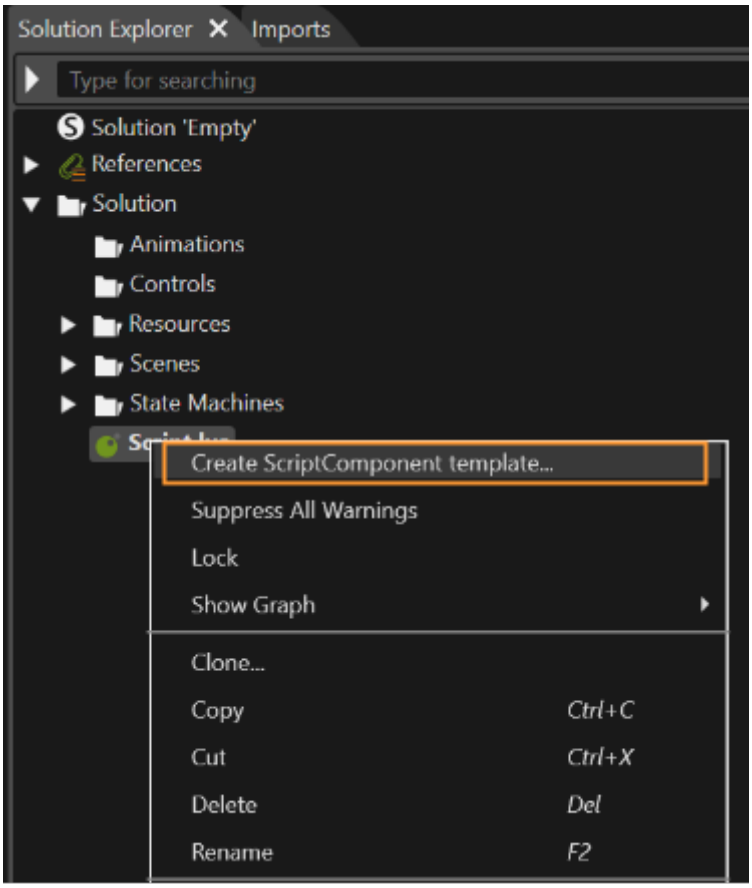


Script Component Templates

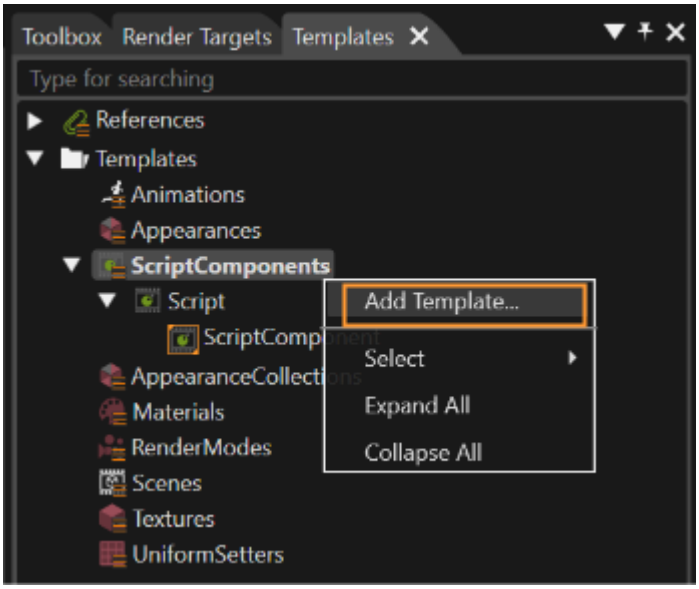
It is possible to create a template from an existing script component. To do this, just right click on an existing script component - in Scene extra-Tree - and from the context menu select the "Save as Template... > ScriptComponent..." option.



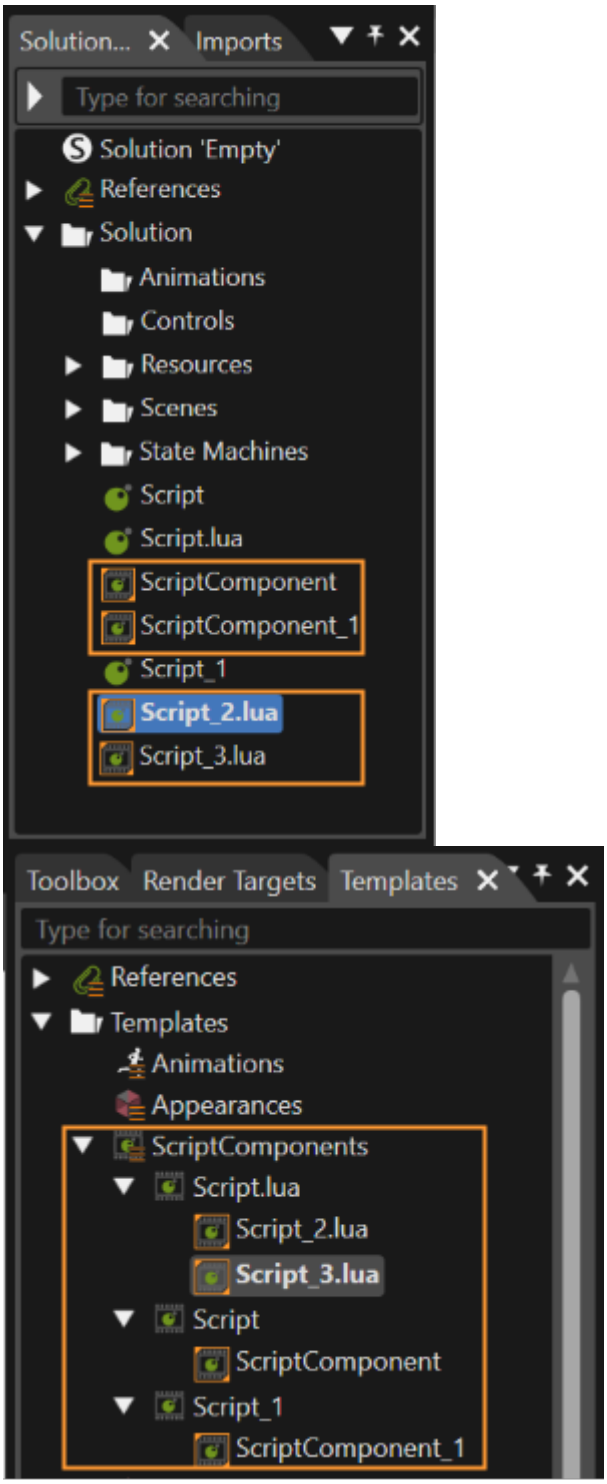
The same operation is possible in "Solution Explorer" panel. Right click on an existing script component and from the context menu select the "Create ScriptComponent Template" option.



The third way to create a template from an existing script component is in the "Templates" panel. Right click on the script component and from the context menu select the "Add Template" option.



In all three cases mentioned above, after the option from the context menu has been selected a dialogue will get opened: "Add New Template". Through this dialogue the user can select a location of the new created template and other details (like Name, Annotations, Script). The created script component templates will be available on both "Solution Explorer" and "Templates" panels.



> If the template is used to create many instances which are set set on different nodes, each will should have a different value for the same parameter if the user set them as such. There should be no reference between the template and its instances.

Post Processing Effects

Until the addition of the predefined effects, the rendering flow of the effect had to be modeled completely with the scene graph including manual management of any temporary render targets involved, creating a rigid structure that was hard to extend and tweak since effect parameters

were scattered around in the scene graph and its assets. That is why the usage of post processing effects was cumbersome and not intuitive. Consequently, the process was simplified and the user has the possibility to use some predefined effects in an easy and intuitive manner.

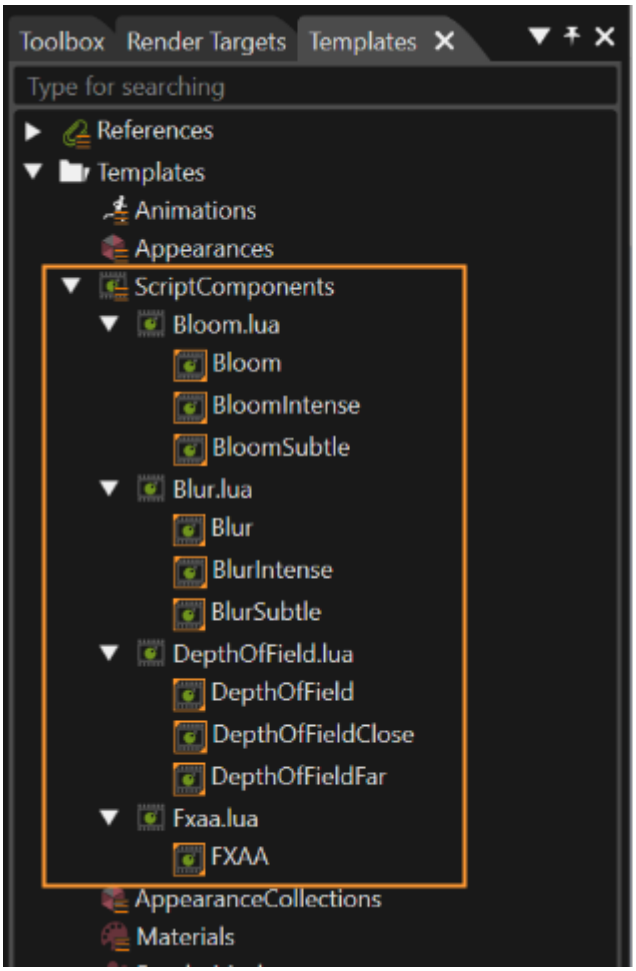
The new scripted post processing pipeline allows the following:

- Drag-and-drop.
- Every script (i.e. effect) has all its parameters exposed in a single UI, and all these parameters can be tweaked in Scene Composer in a WYSIWYG fashion.
- Effects parameter values can be saved as templates. This way the user can save different settings of the same effect/script.
- Effect parameters can be controlled or animated by other scripts or animations, making complex post processing effects possible (rather than just static post effects).
- Several effects on the same camera create a post processing stack where the render target of the camera is automatically passed through to create the final image.
- The order of the elements of that stack can be easily changed via the “Priority” parameter of the script.
- Individual effects in the stack can be enabled and disabled at any time by the user or by another script.

To see how to write a post processing script check the "Candera Introduction" which is available here: Candera Lua Post Processing.

Three predefined post processing effects are available in the "Templates": "References > SCL:ConstructionKit > ScriptComponents". These effects are the following:

- Bloom
- Depth of Field
- FXAA ("Fast Approximate Anti-Aliasing")



To use them, the following steps are mandatory. First, drag some meshes into an already created 3D scene. Second, select one of the available script component templates - "Bloom", "Depth of Field", or "FXAA" - and drop it over the Camera node in the Scene Tree. The camera should be assigned to a render target on a display. Check if the content is visible on the display. Third, start the script system by calling: "Script > Start/Pause" script system menu option or "Ctrl + F6" shortcut.

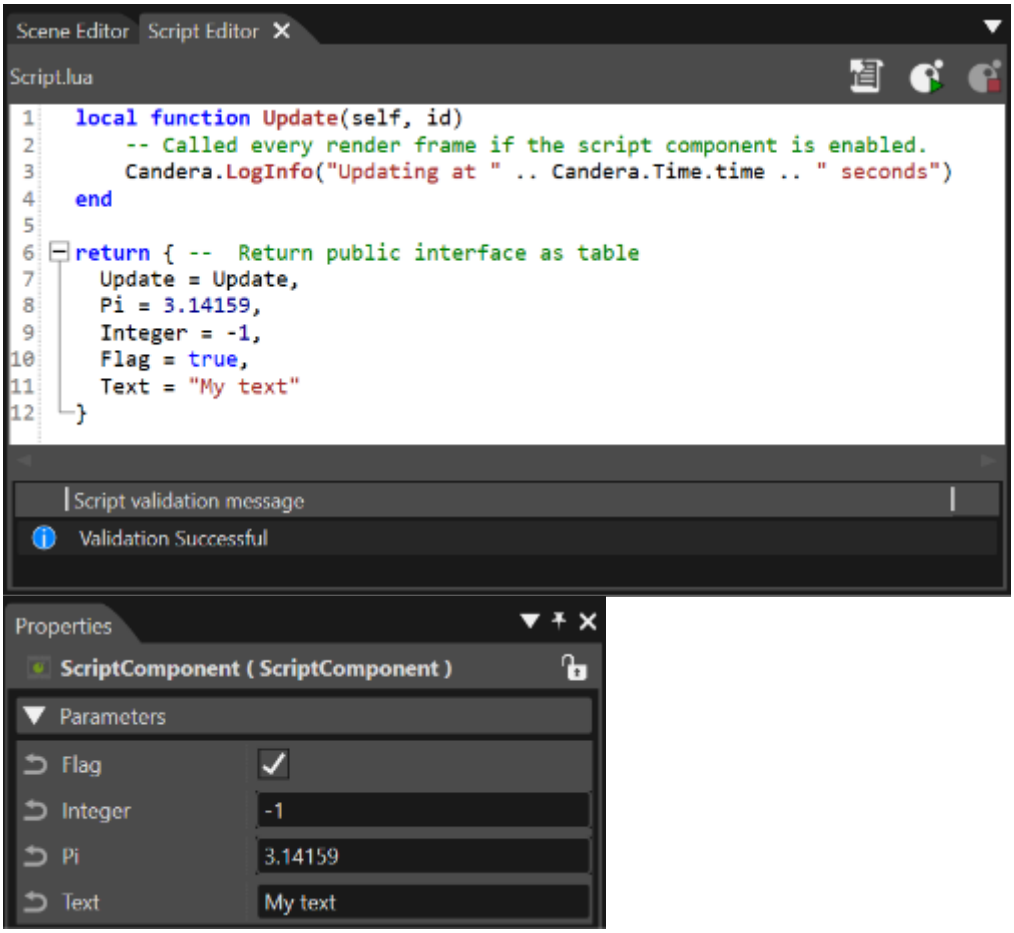
Many details related to Candera scripting can be found here: Candera Lua Object References.

Script Component Parameters

> The terms variable and parameter used in this documentation are interchangeable when it comes to scripting.

Parameters that have been publicly exposed in the script appear automatically in SceneComposer's user interface in the 'Properties' tab of the script component. While the parameters are declared in the script (details about public parameters can be found here), their actual values are defined by the script component. That way the same script can be attached multiple times with different parameter values. This means that scripts can be customized on a per instance (i.e component) basis. Editing and validating the script will not discard the values as long as the parameter has not been removed from the script. The values are saved with the solution and exported as part of the

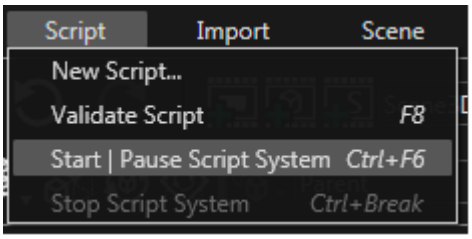
scene.


Scene Editor
Script Editor
Script.lua
local function Update(self, id)
-- Called every render frame if the script component is enabled.
Candera.LogInfo("Updating at " .. Candera.Time.time .. " seconds")
end
return { -- Return public interface as table
Update = Update,
Pi = 3.14159,
Integer = -1,
Flag = true,
Text = "My text"
}
Script validation message
Validation Successful
Properties
ScriptComponent (ScriptComponent)
Parameters
Flag
Integer
-1
Pi
3.14159
Text
My text

Script Execution

Scripts can be executed by selecting "Start Script System" from the Script menu, pressing Ctrl+F6, or clicking the "Start Script System" button in the Script Editor.
Execution of script can be stopped by selecting "Stop Script System" from the Script menu, pressing Ctrl+Break, or clicking the "Stop Script System" button in the Script Editor.

> The operation of starting scripts will validate any script that has not been validated yet.



> To show the Script Editor, go to the Menu bar and select View > Editors > Script Editor.

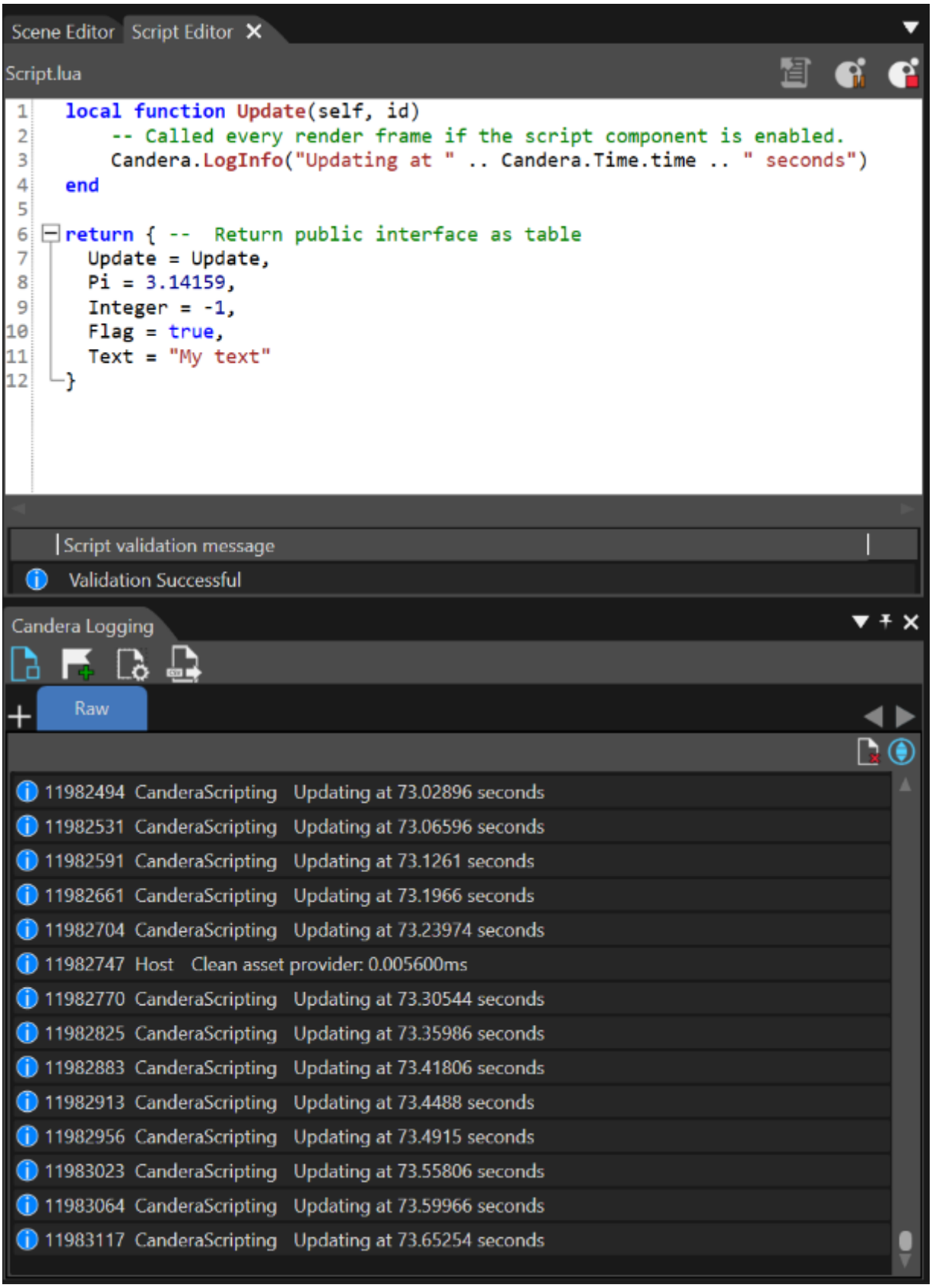


Script System Playing (i.e. executing)

Executing scripts changes the "Start Script System" button in the Script Editor to a "Pause" button:



Executing the script shown in the screenshot produces output in the Candera Logging tab.



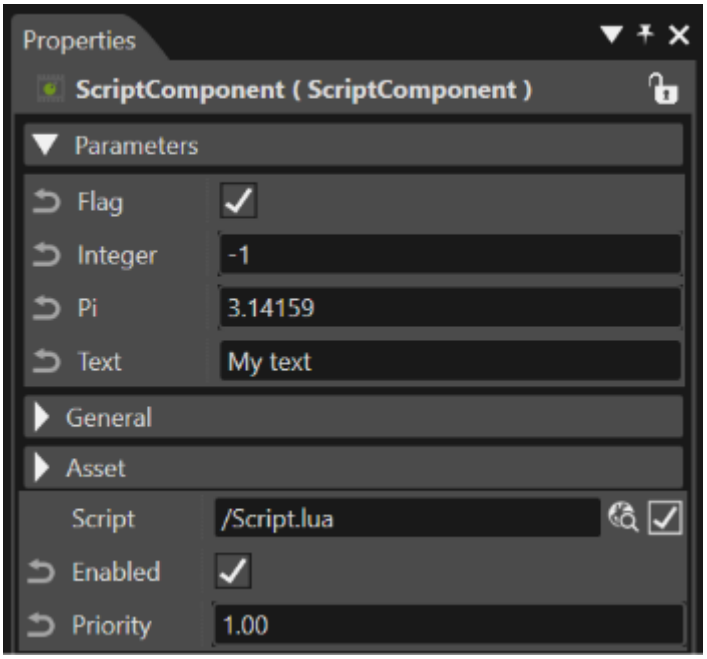
During execution of scripts, their parameters in the ScriptComponent Property tab are updated in realtime to reflect their current values.

Script System Pausing (i.e. execution is paused)

When script execution is paused, the button in the Script Editor changes to "Resume":



When execution of scripts is paused, the parameters in the ScriptComponent Property tab can be edited and will affect script execution once it is resumed. The Enabled flag and the Priority can also be changed in realtime affecting the execution of scripts.



Script System Stopped

When execution is stopped, the parameters of all script components are reset to their values before execution was started. This includes any changes to parameter values during paused execution, as well as the Enabled flag of the component.

> While scripts are being executed, they can be edited in the Script Editor. However they can only be validated when the script system is stopped.

Support for Object Reference Properties

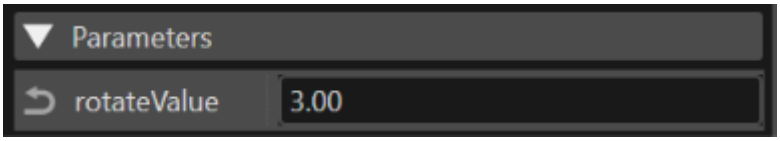
It is possible to assign object reference data types to script properties. Once a script has been assigned to a script component, the property panel of the ScriptComponent is populated with properties exposed by the Script. Dragging an object on a ScriptComponent would set a reference property, if any reference property of the dragged object type is exposed by the ScriptComponent.

> When defining reference parameters in the Script, they become available on every script component that uses that script. Each reference parameters has a tooltip indicating where to attach items from.

Scripts already support a number of property types to be configured in SceneComposer. Candera ScriptSystem has additional support for using CanderaObject references inside scripts.

```
return {
    Init = Init,
    Update = Update,
    rotateValue = 3.0,
    GetName = GetName
    }
```

As can be noticed in the example above, a float property used in a script results in a editable dynamic property:



In a similar way, a reference property should be configurable in the property panel:

```
-- The exposed animation porperty.
WheelAnimation = Candera.CreateReference(Candera.ReferenceType.Animation),
```

shall result in:



Supported References

The following references are supported:

- Animation and AnimationGroups (from solution. control node animations are not supported)
- Appearance and Appearance collections (from the solution. Appearance collection is preferred to have instance sharing enabled). Appearances/Appearance collections attached to nodes are not supported
- 3D camera
- 3D light
- 3D node (any type)
- 2D camera
- 2D node (any type)
- RenderTarget (must be a 3D render target type)

> Reference parameters are not editable when pausing the script (only the value ones). References inside controls are not supported.
