

Installing Custom Nodes

Sometimes you need to install and expose custom patterns or materials for artists. RenderMan for Houdini uses environment variables to point to your custom nodes as well as a couple other mechanisms.

Overview

RenderMan can find your custom nodes easily but Houdini will need to generate the Interface for your nodes. The basic idea is to use RenderMan environment variables to point to the plugins and then run a script provided to generate the Houdini Digital Assets (HDA) that supply the interface. This script can run each time you start Houdini or you can generate the files once and then disable the script to reload these each time you start Houdini to save time. Note that changes to these nodes will require that you re-run the script to update the interface.

Environment Variables

The following RenderMan Environment Variables can be used (either/or) to point to your installation location for your custom nodes, this can be a network location or separated list of multiple locations.

 RMAN_RIXPLUGINPATH
RMAN_SHADERPATH

Typically the RIXPLUGINPATH is meant for C++ plugins and SHADERPATH is meant for OSL and .args files. However, in Houdini it will parse either specified set of locations for all types of files. These can be used inside of your Houdini.env file for convenience if desired.

Creating HDAs

First: Shader names can be used to namespace or version your HDAs. Prefix before the first _ (underscore) will become the namespace and the numerical suffix after the last _ (underscore) will become the version. For example, a shader file called *namespace_myshader_2.osl* will be *namespace::myshader::2* in Houdini. This can allow you to have multiple versions of a shader in your scene and not have to worry about naming your shader the same as a third party asset or Houdini node. See a more detailed explanation of digital asset namespacing and versioning [here](#).

Then, Houdini needs to create HDAs to generate the correct interface for your nodes from the OSL and .args files. A script is provided to accomplish this after parsing the locations specified above.

You can invoke the script using the following environment variable and specifying a *value of 1*.

 RFH_ARGS2HDA = 1

Once the script is run on Houdini startup, it will save the resulting HDA files in the default user directory unless otherwise specified. You can change this location by setting RFH_USER_PREF_DIR. If this isn't set, the HDAs will be saved in HOUDINI_USER_PREF_DIR.

Once these have been created, you can reuse them by resetting the RFH_ARGS2HDA environment variable to 0, otherwise this will run each time you start Houdini and update the nodes automatically. This can be desirable unless there are many nodes causing a slow startup or no updates are being made that need to be captured each startup.

You can find more options for manual operation or possible debugging below, however, the above usage should be enough to accomplish installation of your custom nodes.

Usage: args2hda.py [options] argsfile1 [argsfile2...] This program parses an args file from Pixar's RenderMan and creates hda files.

Outputs

- I hdafile, create a hdafile for a single shader
- L hdafile, add shaders to a library of digital assets (one hda for all shaders)
- d dsfile, create a dialog script for a single shader

Label

- N label, only valid when script called on a single file

Path

- p path, path to find the shader when rendering. This path becomes the shader name property of the hda.

Icon

-C file, icon must already exist in HOUDINI_PATH

-c path, path to icon file to be in embedded in hda

Menu

-m MyCustomNodes

Original Names

-O, keep original names

Version

-V myVersionNumber, adds version to shader

Verbose

-v, output debug messages