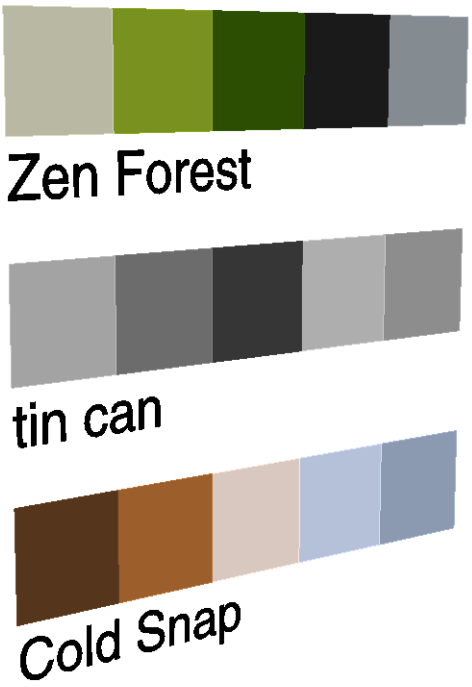
The kuler RSS

—

Accessing a Webservice from a HDA

Georg Duemlein



The Industrial Parametrisation of the World
http://www.preset.de/
2008

Abstract

This document explains concepts and creation digital asset querying a web service for information.
The response is parsed and displayed in the asset's GUI and transformed into a visual representation.

The example used the RSS feed generated by the Adobe® kuler service: http://kuler.adobe.com/

The kuler RSS Digital Asset

The Digital Asset discussed in this document queries a RSS feed and displays the response.
A RSS (Real Simple Syndication) collects information from one or more sources. The generated "feed" contains both human and machine readable information as the format uses tags to classify the data [Wikipedia].

In the case of the kuler RSS the feed contains colour themes generated by users of kuler.adobe.com.
The feed URL is accessible via an API [Adobe] and can be queried in various ways.

The kuler RSS asset allows the user to specify either a colour or a tag. It asks the web service to return a list of themes containing either the colour or the tag. In this example only the first theme will be processed.
The answer is parsed by the asset and used in two ways:

1. The colour swatches of the theme, date of last modification and name of theme and author are displayed in the user interface.
2. The swatches are displayed as a non-renderable object in the scene.

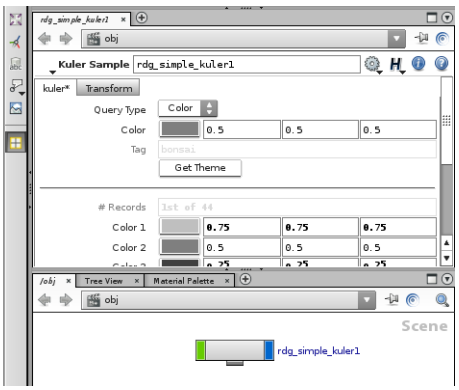


Fig 1. the kuler HDA: Parameter pane and viewport representation

This document doesn't provide a step-by-step tutorial instead we will be exploring the different components of the completed HDA.

Anatomy of the Digital Asset

The Digital Asset consists of different components:

1. the geometric representation
2. the interface
3. the scripts
4. the help and the icon

The geometric representation is the part ultimately displayed in the viewport. In the case of this asset it is the least important part and was build as the final step. Actually the complete asset was developed on the basis of an empty subnetwork at object level. In other assets the geometry can play a much more important role and it's often that we convert a parts of a SOPnetwork into digital asset.

After converting the subnetwork into a digital asset the initial interface layout was designed. In a recursive process the interface and scripts evolved to their final shape.

As soon as the basic functionality was implemented the geometric representation was added to the asset.

The Interface

The interface of the kuler HDA allows the user to specify a colour or a tag.

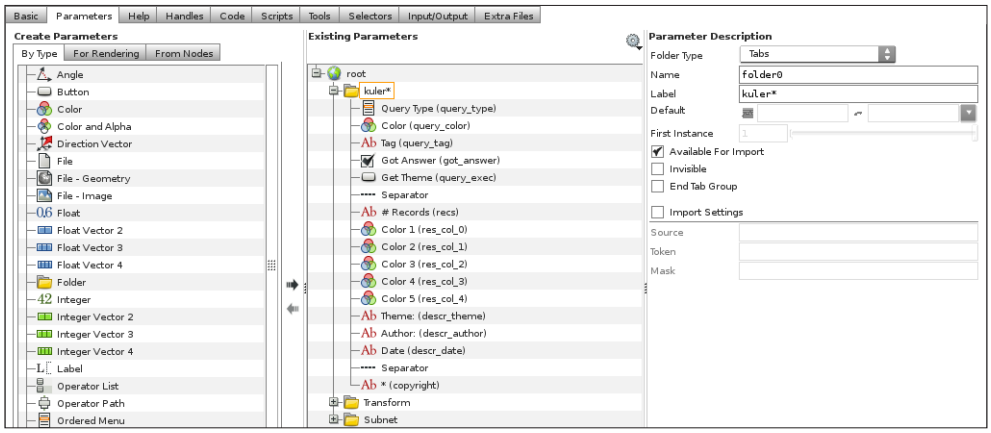


Fig. 2: The "Edit Operator Type Properties" Dialog

The folder *Subnet* was set to invisible as it doesn't contain any useful parameters for this task.
From *Transform* folder all but *Translate, Rotate, Scale, Look At and LookAt Up Vector* were hidden. This allows the user to place the swatches in a prominent position and eventually orientate themselves to the camera.

The *kuler** folder is the main interface for user interaction.

A ordered menu called "Query Type" is used to switch between the two query modes.
The colour parameter and respectively the tag parameter are disabled:

{ query_type == 1}

This expression disables the colour parameter if the menu is set to 'Tag' and returns a 1.

Fig. 3: The input section of the asset's GUI

The second query mode was implemented after the basic functionality was already working.

The toggle "Got Answer" is an invisible parameter and is used internally to control the interface. Other parameters like the response swatches are connected to this toggle. If the web service return no or useless information the swatches are be disabled.

The "Get Theme" button triggers the query.
It executes a function in the the asset's hdaModule:

hou.pwd().hdaModule().onExecQuery()

The hdaModule is a convenient place to store functions for the digital asset. What functions the kuler HDA uses will be explained in the next chapter.

After the separator the response of the web service is displayed.

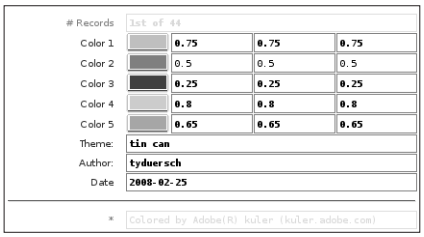


Fig. 4: The input section of the asset's GUI

The *recs* parameter usually displays how many results the service found. In this demonstration only the first record is used. The asset can be extended to handle more themes and switching between them.
If there are no results or an error has occurred the *recs* parameter is used to display an error message.

The five colour parameters are used to display the swatches of the theme.
Theme name, author and date of the last modification are displayed as strings.

The Scripts

The script for this digital asset has been divided in multiple functions.
The design doesn't follow any specific pattern, like Model-View-Controller. For more complex scripts it is highly recommended to use such patterns [Gamma].

The script imports two python modules.

1. urllib2 is used to fetch the RSS feed from the kuler.adobe.com server.
2. xml.dom.minidom is used to parse the response from the server.

```
import urllib2
import xml.dom.minidom as dom
```

kuler expects and returns colour values in hexadecimal notation.
Two auxiliary functions are defined to convert colour values from rgb to hex and vice versa.

```
def rgb2hex(rgb):
    # http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/266466
    hex = '%02x%02x%02x' % rgb
    return hex

def hex2rgb(hex):
    hex = hex.lstrip('#')
    r, g, b = hex[0:2], hex[2:4], hex[4:]
    r, g, b = [int(n, 16) for n in (r, g, b)]
    return (round(r / 255.0, 2), round(g / 255.0, 2), round(b / 255.0, 2))
```

The function *getKulerSwatches(type, query)* handles the communication with the webservice.
First the query string is constructed.
If there is a response from the server the result is parsed. The RSS feed is actually just a XML document. We can manipulate it using standard dom functions:

getElementsByTagName(tag)

This function returns a list of all elements of the specified tag in the document. The kuler API uses a name-space and all tages got the prefix 'kuler:'. The XML structure was analysed in a text editor after saving a response from the server in the web browser.

More important in this function is the error handling.
First we need to check for network errors from urllib2.
Secondly we need to test the server response - is there any record set?

The information is collected into a list and returned to the main function:

```
def onExecQuery():
```

This function is executed if the user pushes the "Get Theme" button in the interface.

The query type is identified and either the colour or the tag prepared for the query.
Type and query are passed to getKulerSwatches().
The query currently isn't urlencoded. Which means that characters like a blank " " will break the communication. We need to check the API documentation how to pass queries to the server.
The response is parsed: Is it a valid theme or an error?
In case of the theme the interface colour swatches will be updated.
In case of an error the error string is displayed.

Sometimes the server responds with a non valid theme.
The tag "smoking" returns no theme though the record count is 2.
Such unexpected behaviour can happen from time to time and needs to be handled by the script.

The Geometry

The geometric representation of this digital asset is rather simple:

A grid of five primitives fetches the colours from the interface:

```
ch("../../resp_col_" + $PR + "r")
```

This example references the red channel from any of the 5 colour parameters in the interface.
`$PR` is substituted by the actual primitive number resulting in on colour per swatch.

A fontSOP references the theme name from the interface:

```
`chs("../../descr_theme")`
```

The expression is wrapped in backticks as the parameter is a string.

The geometry node is set to non-renderable, as it is just a feedback and not meant to be part of the final composition. As a reinsurance the render flag in the SOP itself is set to a nullSOP.

Conclusion

It is harder to write about scripting a HDA than I thought.
One needs to know about Houdini's Digital Assets and about the Python modules required to perform the task. It is highly recommended to outline the interface and the specification sheet before starting to code.
It's easy to get lost.
Houdini's python shell and the system shell should be used to test functions and order of execution.

Georg Duemlein
Februrary 2008

References

Adobe, kuler
http://kuler.adobe.com/ [2/26/2008]

Adobe, kuler Terms of Use
http://kuler.adobe.com/links/kuler_terms_of_use.html [2/26/2008]

Adobe Labs, The kuler API
http://labs.adobe.com/wiki/index.php/Kuler#kuler_APIs [2/26/2008]

Gamma, Erich et. al., Design Patterns. Elements of Reusable Object-Oriented Software, 1995, Addison-Wesley Longman, Amsterdam

Python, urllib2 -- extensible library for opening URLs
http://www.python.org/doc/lib/module-urllib2.html [2/26/2008]

Python, xml.dom.minidom -- Lightweight DOM implementation
http://docs.python.org/dev/3.0/library/xml.dom.minidom.html [2/26/2008]

Wikipedia, RSS
http://en.wikipedia.org/wiki/RSS [2/26/2008]