Tutorial 12: Low Level Field Access with vrFieldAccess

The *Tutorial 12: Low Level Field Access with* `vrFieldAccess` video covers the following:

- Basics (using Node Editor to look up fields) and advanced
- How to access certain more complex fields, such as `fieldContainer` in a field container (and so on, multiple levels) or list of field containers

Download the example scripts here: https://github.com/raemorris/VRED-Python-Tutorial-Examples-Scripts/blob/main/tutorial12.zip?raw=true

Sample Python Code

This is the accompanying example Python script for the *Tutorial 12: Low Level Field Access with* `vrFieldAccess` video.

- To view the video, visit: https://www.youtube.com/watch?v=W3Rhy49lfQc
- To review the video captions, see the *Video Captions* section.

fieldAccess-examples.py

```
# Simple Field Access
material = createMaterial("UPhongMaterial")
material.setName("Python Material")
fields = material.fields()
fields.setVec3f("`diffuseColor`", 1, 0, 0)

# Complex Field Access
leatherMatFields = findMaterial("Leather red").fields()
colorComponentData =
`vrFieldAccess`(leatherMatFields.`getFieldContainer`('colorComponentData')
)
bumpComponent =
`vrFieldAccess`(colorComponentData.`getFieldContainer`('bumpComponent'))
bumpImageComponent =
`vrFieldAccess`(bumpComponent.`getFieldContainer`('image'))
```

```
width = bumpImageComponent.getInt32("width")
height = bumpImageComponent.getInt32("height")

print(width, height)

# Attaching arbitrary data to a node field
envNode = findNode("EnvironmentsTransform")
customAttachment = createAttachment("ValuePair")
`vrFieldAccess`(customAttachment).setMString('key', ['key1', 'key2',
'key3'])
`vrFieldAccess`(customAttachment).setMString('value', ['value1', 'value2',
'value3'])
envNode.addAttachment(customAttachment)

# Reading the values from the attachment
envNode = findNode("EnvironmentsTransform")
attachmentFieldAccess =
`vrFieldAccess`(envNode.getAttachment("ValuePair"))
keys = attachmentFieldAccess.getMString('key')
values = attachmentFieldAccess.getMString('value')

keyValuePairs = list(zip(keys, values))
print(keyValuePairs)
```

Video Captions

Hello and welcome to our final video in our Python scripting tutorials for VRED Pro. My name is Christopher and today I will show you how you can use the `vrFieldAccess` API to read and manipulated fields in VRED's Node Editor.

When you open the Node Editor in VRED via the menu, you have access to all internal data that VRED uses for any type of nodes. This can be, for example, any node that you can have in a Scene Graph, but also material nodes, light nodes, or even sceneplate nodes. If you have a look at a Scene Graph node in the Node Editor, you can see, for example, all the current transform values like translation, rotation, or scaling. For materials, the Node Editor shows all color component data and textures, like diffuse color or bump maps.

Normally, you would not want to access this information via the Node Editor, because VRED already exposes most of this data via the normal interface. Also, it is not safe to edit arbitrary values in the Node Editor, so changing the Perspective Matrix of a camera by hand maybe bad idea.

But, in some cases, it can become handy. For example, if you want to dynamically create materials with Python, the only way to set a diffuse color or other parameters is by using the Field Access API. Another example, for a project, I had to dynamically generate display screens that matched the size of incoming video stream resolutions, and the only way to achieve this was to read out the resolution of the video stream via the Field Access API.

Let's start with a simple example. We want to create a material and set its diffuse color to a bright red. If we have a look at other materials with the Node Editor, we can see that the `diffuseColor` field is responsible for this. When we move our mouse cursor to the right on the value, we also can see the data type that is internally used in VRED. In this case, it's a vector with three components.

For this, we first access the fields of the nodes with `fields()` method and then we use a set method to set the value. In this case, we are using the `getVector3ef` method that lets us write the RGB values to the field `diffuseColor`. When we execute the script and reload the Material Editor, we can find the new material with its diffuse color set to a bright red.

Okay, let's dig deeper. Accessing the first level in the Node Editor was easy. But, how can we reach fields that are deeper in the field access graph. Let's say we want to read the texture resolution of a bump texture of a material. To do this, we first have to get access to each field container that contains the fields we want. In our case, this is "color component data", "bump component", and "image".

We first we our material and its fields. Then, we go from one field container to next, until we reach the field container that actually contains our data. This is done by using the `getFieldContainer` method and using the return value as parameter in the `vrFieldAccess` class constructor. When we have reached our last field container "bump image component", we can access the fields of the resolution. In this case the resolution is 1024 x 1024.

For the most part, you are restricted to only read and manipulate fields that are already present in the node. But, you also can, to some extent, add arbitrary data to a node. This can get useful when developing, for example, complex plugins for data preparation. So, how can we achieve this.

The next script shows how we can use the attachment node "value pair" to write lists of string data to it, and we use the "key" and "value" fields to store our data. We split our key value pairs in two lists, store them in the attachment, and are now able to retrieve them later.

Here, we use the Python zip function to rebuild our original list of key value pairs. But be careful. There are some nodes that already store data in the value pair attachment. So, if you use this, then implement some measures to make sure you don't accidentally change this data.

A quick note on the data types and how we make sure we are using the correct data type. Sometimes it's not clear if an integer is 8-bit, 16-bit, signed, or unsigned. If you never dealt with C or C++ you might not even know what to do with it. You can always hover above the value field in the Node Editor to see the actual internal data type that is used. This gives you a hint on what method you have to use to write or read the values from the fields. What I like to do is to do a full text search in the documentation and try to find matching method for the data type in the Node Editor. If you take the wrong method, you get an error message. So, just make sure to test the method you are using, and you should be fine.

That’s it for today! Thank you for joining me. This was the last video of our Python scripting tutorial series for VRED Pro. But we also have Python tutorial for the new VRED Core if you are interested. So, see you there.