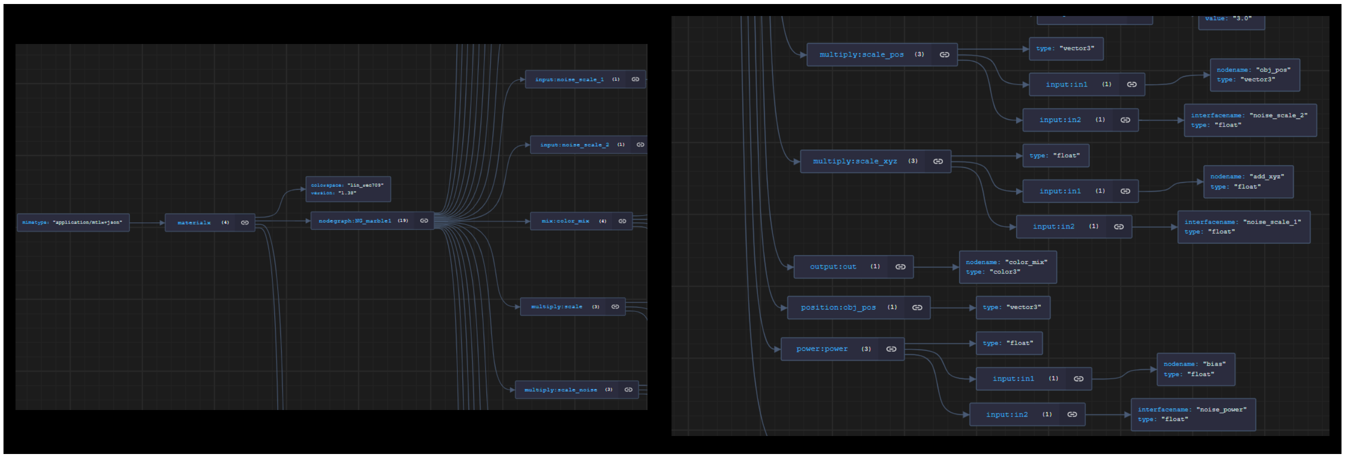
MaterialX XML / JSON Notebook

In this notebook we will look how JSON can be used in the context of MaterialX interop. Items to examine include:

- Extracting out "interfaces" for `nodegraph` s
- Converting XML to JSON.
- Determining a JSON schema.
- Validating JSON data against such as schema



Snaphots of graphs generated using the JSONCrack add-on.

Setup for JSON support

We will use the `xmltodict` Python package to convert from MaterialX represented in XML to JSON. The JSON package `json` will then be used to manipulate data.

```
In [ ]: import MaterialX as mx
        import mtlxutils.mxfile as mxf

        import xmltodict
        import json

        from IPython.display import display_markdown

        # Print xmltodict version
        print('xmltodict version: ', xmltodict.__version__)
        print('json version: ', json.__version__)
        print('materialx version: ', mx.__version__)
```

```
xmltodict version:  0.13.0
json version:  2.0.9
materialx version:  1.38.8
```

Nodegraph Interface Extraction

We are interested in just the "pattern graphs" which are connected to any surface shader connected to a material. We define two utility functions:

- `getShaderNodes()` : To get all the surface shader nodes either connected to a material or not.
- `getRenderableGraphs` : To find any upstream graphs connected to a surface shader.

In [ ]:

```
def getShaderNodes(graphElement):
    '''
    Find all surface shaders in a GraphElement.
    '''
    shaderNodes = set()
    for material in graphElement.getMaterialNodes():
        for shader in mx.getShaderNodes(material):
            shaderNodes.add(shader.getNamePath())
    for shader in graphElement.getNodes():
        if shader.getType() == 'surfaceshader':
            shaderNodes.add(shader.getNamePath())
    return shaderNodes


def getRenderableGraphs(graphElement):
    '''
    Find all renderable graphs in a GraphElement.
    '''
    ngnamepaths = set()
    graphs = []
    shaderNodes = getShaderNodes(graphElement)
    for shaderPath in shaderNodes:
        shader = doc.getDescendant(shaderPath)
        for input in shader.getInputs():
            ngString = input.getNodeGraphString()
            if ngString and ngString not in ngnamepaths:
                graphs.append(graphElement.getNodeGraph(ngString))
                ngnamepaths.add(ngString)
    return graphs
```

For each graph, we only wish to include the interface and include any connections on inputs or outputs. This provides a clean encapsulation of such a graph.

All graphs are assumed to be parented directly under the root Document.

Currently the process to just copy and extract this nodegraphs is overtly complex if we just want the interace. The `copyContentsFrom()` interface for nodegraphs copies over too much information requiring the removal of unneeded children. The alternative is to manually copy over nodegraph attributes, and child inputs and outputs.

It would be useful to have the logic encapsulated in a single API call.

We add in a `copyGraphInterfaces()` function which will copy only the interface of a `nodegraph` to a new `nodegraph` under a given document element.

In [ ]:

```
def copyGraphInterfaces(dest, ng):
    '''
    Copy the interface of a nodegraph to a new nodegraph under a specified parent `dest`.
    '''
    copyMethod = 'add_remove'
    ng1 = dest.addNodeGraph(ng.getName())
    if copyMethod == 'add_remove':
        ng1.copyContentFrom(ng)
        for child in ng1.getChildren():
            if child.isA(mx.Input) or child.isA(mx.Output):
                for attr in ['nodegraph', 'nodename', 'defaultinput']:
                    child.removeAttribute(attr)
                continue
            ng1.removeChild(child.getName())
    else:
        for attrName in ng.getAttributeNames():
            attr = ng.getAttribute(attrName)
            newattr = ng1.addAttribute(attr.getName(), attr.getType(), attr.getValue())
            newattr.copyContentFrom(attr)
        for port in ng.getInputs():
            newport = ng1.addInput(port.getName(), port.getType())
            newport.copyContentFrom(port)
        for port in ng.getOutputs():
            newport = ng1.addOutput(port.getName(), port.getType())
            newport.copyContentFrom(port)
            for attr in ['nodegraph', 'nodename', 'defaultinput']:
                newport.removeAttribute(attr)
```

The top level logic, loads in the "Brick" graph which can be found in the Examples folder of a MaterialX distribution. These are extracted out to a new document from which we get the representation as an XML string. The results are printed out.

In [ ]:

```
# Load in sample MaterialX file
doc, libFiles, status = mxf.MtlxFile.createWorkingDocument()
mx.readFromXmlFile(doc, './data/standard_surface_brick_procedural.mtlx')

# Create destination document and copy nodegraph interfaces over
xmldoc = mx.createDocument()
graphs = getRenderableGraphs(doc)
for ng in graphs:
    copyGraphInterfaces(xmldoc, ng)

# Get interfaces as an XML string
xml_string = mxf.MtlxFile.writeDocumentToString(xmldoc)

text = '<details><summary>Extracted MaterialX Nodegraphs in XML</summary>\n\n' + '```xml\n' + xm
display_markdown(text, raw=True)
```

▼ Extracted MaterialX Nodegraphs in XML

```xml
<?xml version="1.0"?>
<materialx version="1.38">
  <nodegraph name="NG_BrickPattern">
    <input name="brick_color" type="color3" value="0.661876, 0.19088, 0" uiname="Brick Color" uifolder="Color" />
    <input name="hue_variation" type="float" value="0.083" uimin="0" uimax="1" uiname="Hue Variation" uifolder="Color" />
    <input name="value_variation" type="float" value="0.787" uimin="0" uimax="1" uiname="Value Variation" uifolder="Color" />
    <input name="roughness_amount" type="float" value="0.853" uimin="0" uimax="1" uiname="Roughness Amount" uifolder="Roughness" />
    <input name="dirt_color" type="color3" value="0.56372, 0.56372, 0.56372" uiname="Dirt Color" uifolder="Dirt" />
    <input name="dirt_amount" type="float" value="0.248" uimin="0" uimax="1" uiname="Dirt Amount" uifolder="Dirt" />
    <input name="uvtiling" type="float" value="3" uisoftmin="1" uisoftmax="16" uiname="UV Tiling" uifolder="Texturing" />
    <output name="base_color_output" type="color3" />
    <output name="specular_roughness_output" type="float" />
    <output name="normal_output" type="vector3" />
  </nodegraph>
</materialx>
```

Test : XML text to JSON

As a first test we will first just convert the XML string to JSON.

After this extraction process we parse the `XML` string to produce the equivalent data as a `Python` dictionary using the `xmltodict.parse()` interfaces.

This is then converted into a string to get the required `JSON` data using e `json.dumps`. (For the sake of display formatting some additional indentation is specified when dumping out a string.)

In [ ]:

```python
# Build JSON from XML
options = {
    'attr_prefix': '',           # Set prefix for attributes
}
python_dict = xmltodict.parse(xml_string, **options)
json_string = json.dumps(python_dict)
json_string_fmt = json.dumps(python_dict, indent=2)

text = '<details><summary>MaterialX in JSON</summary>\n\n' + '```json\n' + json_string_fmt + '\n
display_markdown(text, raw=True)

with open('mtlx_brick.json', 'w') as jsonfile:
    jsonfile.write(json_string_fmt)
```

▼ MaterialX in JSON

```
{
  "materialx": {
    "version": "1.38",
    "nodegraph": {
      "name": "NG_BrickPattern",
      "input": [
        {
          "name": "brick_color",
          "type": "color3",
          "value": "0.661876, 0.19088, 0",
          "uiname": "Brick Color",
          "uifolder": "Color"
        },
        {
          "name": "hue_variation",
          "type": "float",
          "value": "0.083",
          "uimin": "0",
          "uimax": "1",
          "uiname": "Hue Variation",
          "uifolder": "Color"
        },
        {
          "name": "value_variation",
          "type": "float",
          "value": "0.787",
          "uimin": "0",
          "uimax": "1",
          "uiname": "Value Variation",
          "uifolder": "Color"
        },
        {
          "name": "roughness_amount",
          "type": "float",
          "value": "0.853",
          "uimin": "0",
          "uimax": "1",
          "uiname": "Roughness Amount",
          "uifolder": "Roughness"
        },
        {
          "name": "dirt_color",
          "type": "color3",
          "value": "0.56372, 0.56372, 0.56372",
          "uiname": "Dirt Color",
          "uifolder": "Dirt"
        },
        {
          "name": "dirt_amount",
          "type": "float",
          "value": "0.248",
          "uimin": "0",
          "uimax": "1",
          "uiname": "Dirt Amount",
          "uifolder": "Dirt"
        },
        {
          "name": "uvtiling",
          "type": "float",
          "value": "3",
          "uisoftmin": "1",
          "uisoftmax": "16",
          "uiname": "UV Tiling",
          "uifolder": "Texturing"
        }
      ],
      "output": [
        {
          "name": "base_color_output",
          "type": "color3"
        },
        {
          "name": "specular_roughness_output",
          "type": "float"
        },
        {
          "name": "normal_output",
          "type": "vector3"
        }
      ]
    }
  }
}
```

Explicit Conversion

The desire is to introduce a standardized JSON representation for MaterialX. For this a match for what is supported for XML is required.

XML is supported via the `MaterialXFormat` library. This this JSON can be added.

For this book, we will create a bidirectional conversion between XML and JSON. Thus instead of a non-standard conversion using a generic converter like `xmltodict` we will the logic in these scripts (and eventually the `MaterialXFormat` library) for JSON conversion.

Some key factors to consider include:

1. The JSON representation should be a direct match to the XML representation.
2. The JSON representation should match the XML size as closely as possible.
3. Export and import options for XML should be supported for JSON.
4. There is no concept of "includes" in JSON. This is a concept specific to XML.

JSON Serialization

To perform a proper serialization, the MaterialX document itself should be examined with direct conversion to JSON.

Serialization to JSON

For conversion to JSON we introduce two functions:

1. `documentToJSON()` : Converts a MaterialX document to a JSON string.
2. `elementToJSON()` : Converts a MaterialX element to a JSON string, and continues to recursively convert any children.

In [ ]:

```python
# We use a colon to separate the category and name of an element in the JSON hierarchy
JSON_CATEGORY_NAME_SEPARATOR = ':'
# The root of the JSON hierarchy
MATERIALX_DOCUMENT_ROOT = 'materialx'

# Convert MaterialX element to JSON
def elementToJSON(elem, jsonParent):
    '''
    Convert an MaterialX XML element to JSON.
    Will recursively traverse the parent/child Element hierarchy.
    '''
    if (elem.getSourceUri() != ""):
        return

    # Create a new JSON element for the MaterialX element
    jsonElem = {}

    # Add attributes
    for attrName in elem.getAttributeNames():
        jsonElem[attrName] = elem.getAttribute(attrName)

    # Add children
    for child in elem.getChildren():
        jsonElem = elementToJSON(child, jsonElem)

    # Add element to parent
    jsonParent[elem.getCategory() + JSON_CATEGORY_NAME_SEPARATOR + elem.getName()] = jsonElem
    return jsonParent

# Convert MaterialX document to JSON
def documentToJSON(doc):
    '''Convert an MaterialX XML document to JSON'''
    root = {}
    root["materialx"] = {}
    for attrName in doc.getAttributeNames():
        root[attrName] =  doc.getAttribute(attrName)

    for elem in doc.getChildren():
        elementToJSON(elem, root[MATERIALX_DOCUMENT_ROOT])

    result = json.dumps(root, indent=2)
    return result
```

We call `documentToJSON()` to convert both the entire document as well as just the NodeGraph interface document below:

In [ ]:

```
# Convert entire document
doc_result = documentToJSON(doc)

text = '<details><summary>Entire document to JSON</summary>\n\n' + '```json\n' + doc_result  + '
display_markdown(text, raw=True)

# Convert just the graph
graph_result = documentToJSON(xmldoc)

text = '<details><summary>Node Graph Interface to JSON</summary>\n\n' + '```json\n' + graph_resu
display_markdown(text, raw=True)
```

▼ Entire document to JSON

```
{
  "materialx": {
    "nodegraph:NG_BrickPattern": {
      "input:brick_color": {
        "type": "color3",
        "value": "0.661876, 0.19088, 0",
        "uiname": "Brick Color",
        "uifolder": "Color"
      },
      "input:hue_variation": {
        "type": "float",
        "value": "0.083",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Hue Variation",
        "uifolder": "Color"
      },
      "input:value_variation": {
        "type": "float",
        "value": "0.787",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Value Variation",
        "uifolder": "Color"
      },
      "input:roughness_amount": {
        "type": "float",
        "value": "0.853",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Roughness Amount",
        "uifolder": "Roughness"
      },
      "input:dirt_color": {
        "type": "color3",
        "value": "0.56372, 0.56372, 0.56372",
        "uiname": "Dirt Color",
        "uifolder": "Dirt"
      },
      "input:dirt_amount": {
        "type": "float",
        "value": "0.248",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Dirt Amount",
        "uifolder": "Dirt"
```

```
},
"input:uvtiling": {
  "type": "float",
  "value": "3",
  "uisoftmin": "1",
  "uisoftmax": "16",
  "uiname": "UV Tiling",
  "uifolder": "Texturing"
},
"multiply:node_multiply_5": {
  "type": "color3",
  "input:in1": {
    "type": "color3",
    "nodename": "node_mix_6"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_tiledimage_float_7"
  }
},
"mix:node_mix_8": {
  "type": "color3",
  "input:fg": {
    "type": "color3",
    "nodename": "node_multiply_5"
  },
  "input:bg": {
    "type": "color3",
    "nodename": "node_multiply_9"
  },
  "input:mix": {
    "type": "float",
    "nodename": "node_tiledimage_float_10"
  }
},
"constant:node_color_11": {
  "type": "color3",
  "input:value": {
    "type": "color3",
    "value": "0.263273, 0.263273, 0.263273"
  }
},
"multiply:node_multiply_9": {
  "type": "color3",
  "input:in1": {
    "type": "color3",
    "nodename": "node_color_11"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_tiledimage_float_7"
  }
},
"rgbtohsv:node_rgbtohsv_12": {
  "type": "color3",
  "input:in": {
    "type": "color3",
    "interfacename": "brick_color"
  }
},
"combine3:node_combine3_color3_13": {
  "type": "color3",
  "input:in1": {
    "type": "float",
    "nodename": "node_multiply_14"
  },
  "input:in2": {
    "type": "float",
    "value": "0"
  },
  "input:in3": {
    "type": "float",
    "nodename": "node_multiply_15"
  }
},
"add:node_add_16": {
  "type": "color3",
  "input:in1": {
    "type": "color3",
    "nodename": "node_combine3_color3_13"
  },
  "input:in2": {
    "type": "color3",
    "nodename": "node_rgbtohsv_12"
  }
},
"hsvtorgb:node_hsvtorgb_17": {
  "type": "color3",
  "input:in": {
    "type": "color3",
    "nodename": "node_add_16"
  }
},
"subtract:node_subtract_18": {
      "type": "float",
      "input:in1": {
        "type": "float",
        "nodename": "node_add_19"
      },
      "input:in2": {
        "type": "float",
        "value": "0.35"
      }
    },
    "multiply:node_multiply_14": {
      "type": "float",
      "input:in1": {
        "type": "float",
        "nodename": "node_subtract_18"
      },
      "input:in2": {
        "type": "float",
        "interfacename": "hue_variation"
      }
    },
    "multiply:node_multiply_15": {
      "type": "float",
      "input:in1": {
        "type": "float",
        "nodename": "node_add_19"
      },
      "input:in2": {
        "type": "float",
        "nodename": "node_multiply_20"
      }
    },
    "clamp:node_clamp_0": {
      "type": "color3",
      "input:in": {
        "type": "color3",
        "nodename": "node_mix_8"
      }
    },
    "multiply:node_multiply_1": {
      "type": "float",
      "input:in1": {
        "type": "float",
        "nodename": "node_divide_21"
      },
      "input:in2": {
        "type": "float",
    "nodename": "node_tiledimage_float_22"
  }
},
"max:node_max_1": {
  "type": "float",
  "input:in1": {
    "type": "float",
    "nodename": "node_tiledimage_float_10"
  },
  "input:in2": {
    "type": "float",
    "value": "0.00001"
  }
},
"divide:node_divide_21": {
  "type": "float",
  "input:in1": {
    "type": "float",
    "interfacename": "roughness_amount"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_max_1"
  }
},
"mix:node_mix_6": {
  "type": "color3",
  "input:fg": {
    "type": "color3",
    "interfacename": "dirt_color"
  },
  "input:bg": {
    "type": "color3",
    "nodename": "node_hsvtorgb_17"
  },
  "input:mix": {
    "type": "float",
    "nodename": "node_multiply_23"
  }
},
"multiply:node_multiply_23": {
  "type": "float",
  "input:in1": {
    "type": "float",
    "interfacename": "dirt_amount"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_tiledimage_float_24"
  }
},
"multiply:node_multiply_25": {
  "type": "float",
  "input:in1": {
    "type": "float",
    "interfacename": "hue_variation"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_tiledimage_float_26"
  }
},
"add:node_add_19": {
  "type": "float",
  "input:in1": {
    "type": "float",
    "nodename": "node_multiply_25"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_tiledimage_float_7"
  }
},
"multiply:node_multiply_20": {
  "type": "float",
  "input:in1": {
    "type": "float",
    "interfacename": "value_variation"
  },
  "input:in2": {
    "type": "float",
    "nodename": "node_tiledimage_float_26"
  }
},
"normalmap:node_normalmap_3": {
  "type": "vector3",
  "input:in": {
    "type": "vector3",
    "nodename": "node_tiledimage_vector3_27"
  }
},
"convert:node_convert_1": {
  "type": "vector2",
  "input:in": {
    "type": "float",
    "interfacename": "uvtiling"
  }
},
"tiledimage:node_tiledimage_vector3_27": {
  "type": "vector3",
  "input:file": {
    "type": "filename",
    "value": "brick_normal.jpg"
  },
  "input:uvtiling": {
    "type": "vector2",
    "nodename": "node_convert_1"
  }
},
"tiledimage:node_tiledimage_float_22": {
  "type": "float",
  "input:file": {
    "type": "filename",
    "value": "brick_roughness.jpg"
  },
  "input:uvtiling": {
    "type": "vector2",
    "nodename": "node_convert_1"
  }
},
"tiledimage:node_tiledimage_float_10": {
  "type": "float",
  "input:file": {
    "type": "filename",
    "value": "brick_mask.jpg"
  },
  "input:uvtiling": {
    "type": "vector2",
    "nodename": "node_convert_1"
  }
},
"tiledimage:node_tiledimage_float_7": {
  "type": "float",
  "input:file": {
    "type": "filename",
    "value": "brick_base_gray.jpg"
  },
  "input:uvtiling": {
    "type": "vector2",
    "nodename": "node_convert_1"
  }
```

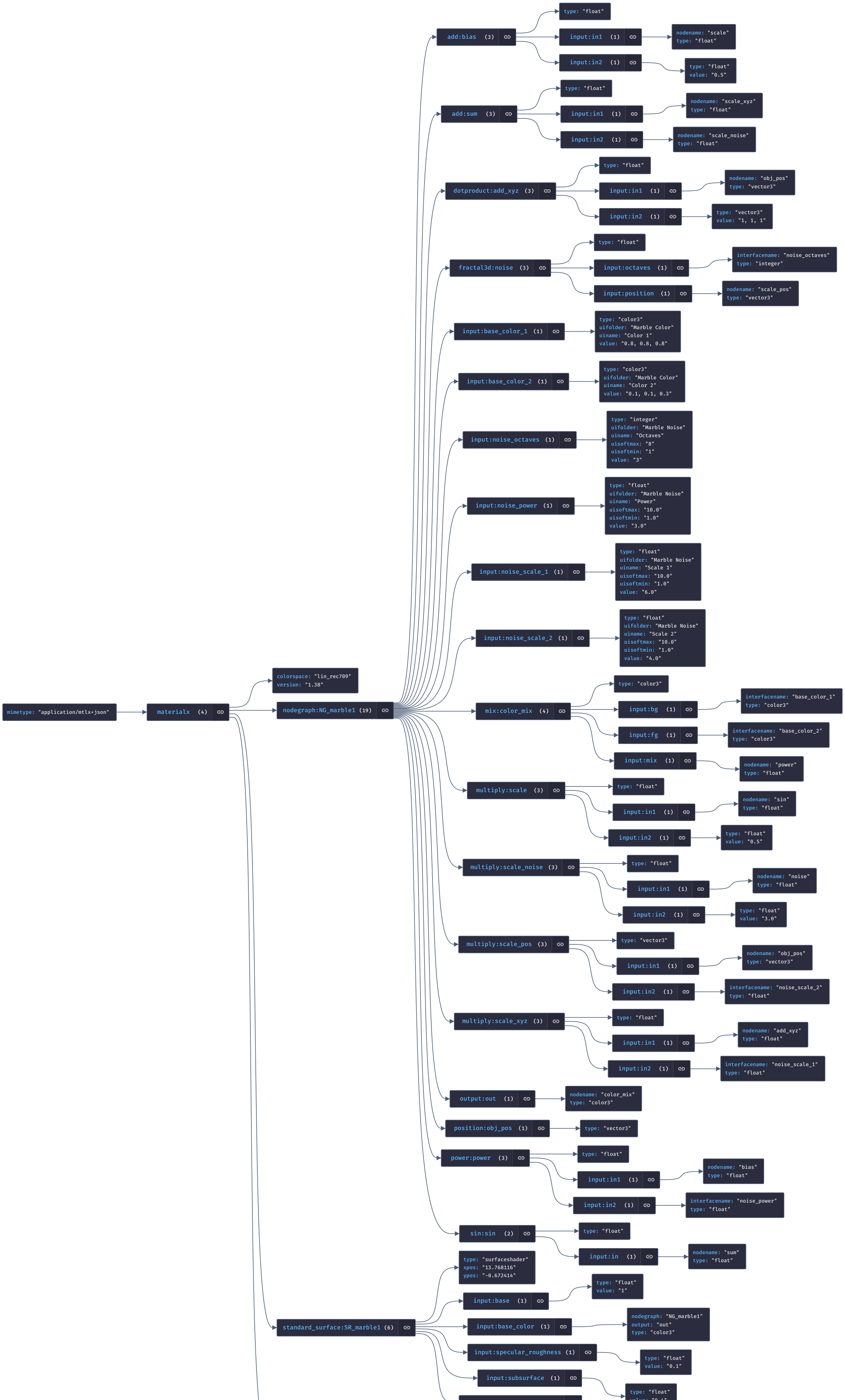
```
      },
      "tiledimage:node_tiledimage_float_26": {
        "type": "float",
        "input:file": {
          "type": "filename",
          "value": "brick_variation_mask.jpg"
        },
        "input:uvtiling": {
          "type": "vector2",
          "nodename": "node_convert_1"
        }
      },
      "tiledimage:node_tiledimage_float_24": {
        "type": "float",
        "input:file": {
          "type": "filename",
          "value": "brick_dirt_mask.jpg"
        },
        "input:uvtiling": {
          "type": "vector2",
          "nodename": "node_convert_1"
        }
      },
      "output:base_color_output": {
        "type": "color3",
        "nodename": "node_clamp_0"
      },
      "output:specular_roughness_output": {
        "type": "float",
        "nodename": "node_multiply_1"
      },
      "output:normal_output": {
        "type": "vector3",
        "nodename": "node_normalmap_3"
      }
    },
    "standard_surface:N_StandardSurface": {
      "type": "surfaceshader",
      "input:base_color": {
        "type": "color3",
        "nodegraph": "NG_BrickPattern",
        "output": "base_color_output"
      },
      "input:specular_roughness": {
        "type": "float",
        "nodegraph": "NG_BrickPattern",
        "output": "specular_roughness_output"
      },
      "input:normal": {
        "type": "vector3",
        "nodegraph": "NG_BrickPattern",
        "output": "normal_output"
      }
    },
    "surfacematerial:M_BrickPattern": {
      "type": "material",
      "input:surfaceshader": {
        "type": "surfaceshader",
        "nodename": "N_StandardSurface"
      }
    }
  },
  "version": "1.38",
  "colorspace": "lin_rec709",
  "fileprefix": "../../../Images/"
}
```

▼ Node Graph Interface to JSON

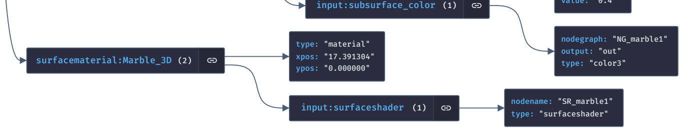
```
{
  "materialx": {
    "nodegraph:NG_BrickPattern": {
      "input:brick_color": {
        "type": "color3",
        "value": "0.661876, 0.19088, 0",
        "uiname": "Brick Color",
        "uifolder": "Color"
      },
      "input:hue_variation": {
        "type": "float",
        "value": "0.083",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Hue Variation",
        "uifolder": "Color"
      },
      "input:value_variation": {
        "type": "float",
        "value": "0.787",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Value Variation",
        "uifolder": "Color"
      },
      "input:roughness_amount": {
        "type": "float",
        "value": "0.853",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Roughness Amount",
        "uifolder": "Roughness"
      },
      "input:dirt_color": {
        "type": "color3",
        "value": "0.56372, 0.56372, 0.56372",
        "uiname": "Dirt Color",
        "uifolder": "Dirt"
      },
      "input:dirt_amount": {
        "type": "float",
        "value": "0.248",
        "uimin": "0",
        "uimax": "1",
        "uiname": "Dirt Amount",
        "uifolder": "Dirt"
      },
      "input:uvtiling": {
        "type": "float",
        "value": "3",
        "uisoftmin": "1",
        "uisoftmax": "16",
        "uiname": "UV Tiling",
        "uifolder": "Texturing"
      },
      "output:base_color_output": {
        "type": "color3"
      },
      "output:specular_roughness_output": {
        "type": "float"
      },
      "output:normal_output": {
        "type": "vector3"
      }
    }
  },
  "version": "1.38"
}
```

The JSON is visualized as a graph for the entire document below, as well as another sample conversion of the glTF "Boombox" and "Olvies" examples from the sample models library

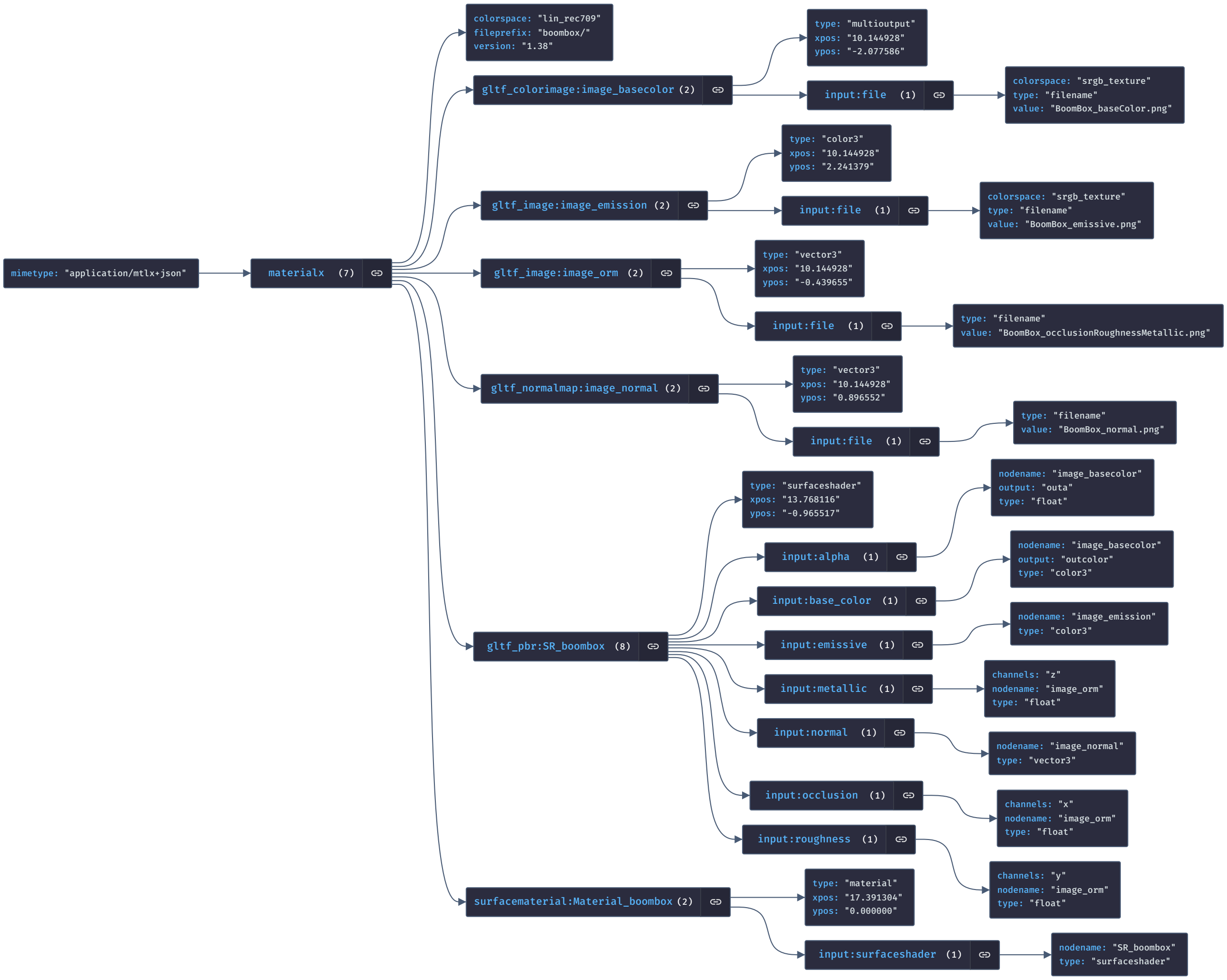
▼ Graph of Marble Material



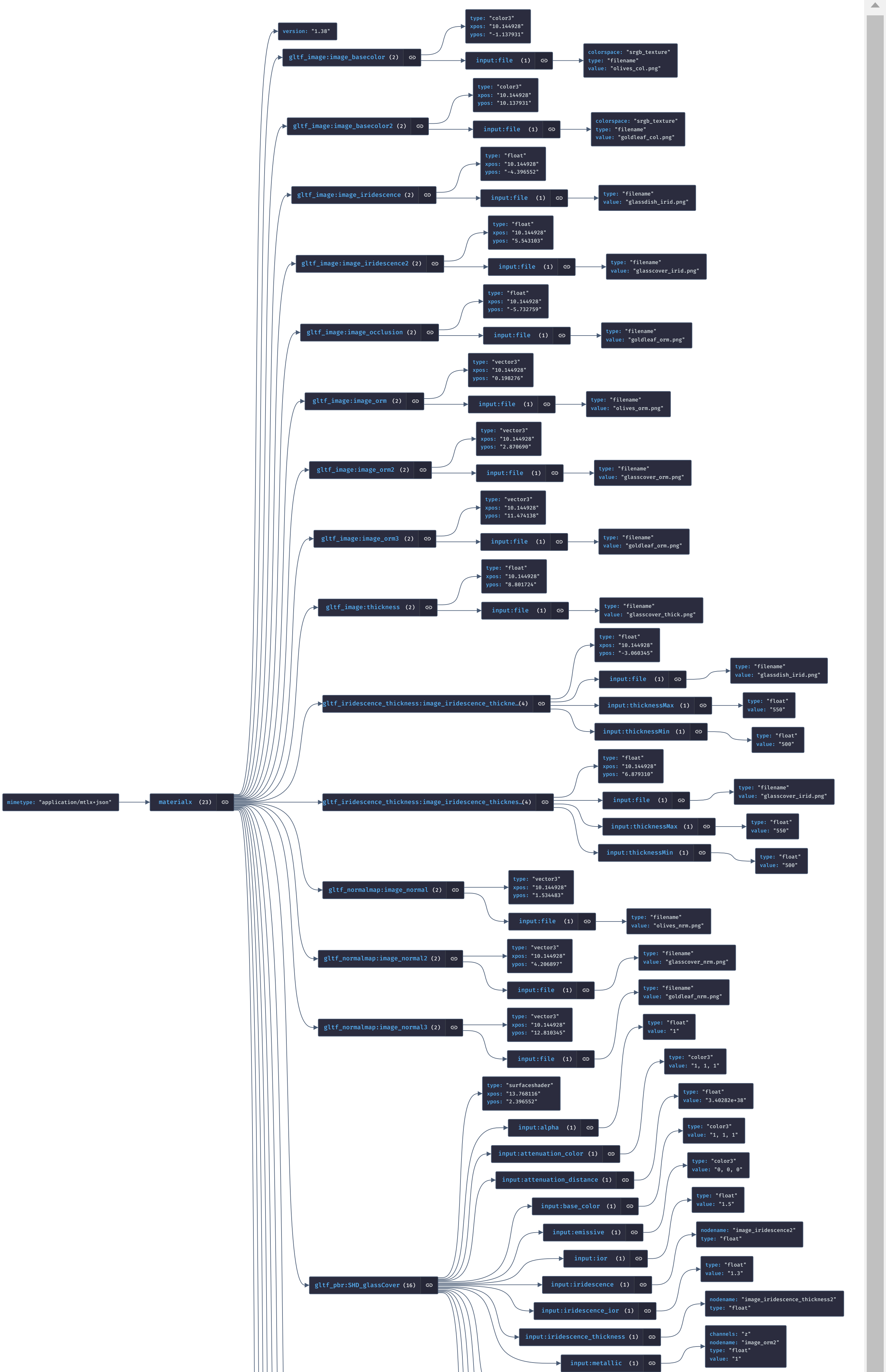
```
mimetype: "application/mtlx+json"
materialx (4)
  colorspace: "lin_rec709"
  version: "1.38"
  nodegraph:NG_marble1 (19)
    add:bias (3)
      type: "float"
      input:in1 (1)
        nodename: "scale"
        type: "float"
      input:in2 (1)
        type: "float"
        value: "0.5"
    add:sum (3)
      type: "float"
      input:in1 (1)
        nodename: "scale_xyz"
        type: "float"
      input:in2 (1)
        nodename: "scale_noise"
        type: "float"
    dotproduct:add_xyz (3)
      type: "float"
      input:in1 (1)
        nodename: "obj_pos"
        type: "vector3"
      input:in2 (1)
        type: "vector3"
        value: "1, 1, 1"
    fractal3d:noise (3)
      type: "float"
      input:octaves (1)
        interfacename: "noise_octaves"
        type: "integer"
      input:position (1)
        nodename: "scale_pos"
        type: "vector3"
    input:base_color_1 (1)
      type: "color3"
      uifolder: "Marble Color"
      uiname: "Color 1"
      value: "0.8, 0.8, 0.8"
    input:base_color_2 (1)
      type: "color3"
      uifolder: "Marble Color"
      uiname: "Color 2"
      value: "0.1, 0.1, 0.3"
    input:noise_octaves (1)
      type: "integer"
      uifolder: "Marble Noise"
      uiname: "Octaves"
      uisoftmax: "8"
      uisoftmin: "1"
      value: "3"
    input:noise_power (1)
      type: "float"
      uifolder: "Marble Noise"
      uiname: "Power"
      uisoftmax: "10.0"
      uisoftmin: "1.0"
      value: "3.0"
    input:noise_scale_1 (1)
      type: "float"
      uifolder: "Marble Noise"
      uiname: "Scale 1"
      uisoftmax: "10.0"
      uisoftmin: "1.0"
      value: "6.0"
    input:noise_scale_2 (1)
      type: "float"
      uifolder: "Marble Noise"
      uiname: "Scale 2"
      uisoftmax: "10.0"
      uisoftmin: "1.0"
      value: "4.0"
    mix:color_mix (4)
      type: "color3"
      input:bg (1)
        interfacename: "base_color_1"
        type: "color3"
      input:fg (1)
        interfacename: "base_color_2"
        type: "color3"
      input:mix (1)
        nodename: "power"
        type: "float"
    multiply:scale (3)
      type: "float"
      input:in1 (1)
        nodename: "sin"
        type: "float"
      input:in2 (1)
        type: "float"
        value: "0.5"
    multiply:scale_noise (3)
      type: "float"
      input:in1 (1)
        nodename: "noise"
        type: "float"
      input:in2 (1)
        type: "float"
        value: "3.0"
    multiply:scale_pos (3)
      type: "vector3"
      input:in1 (1)
        nodename: "obj_pos"
        type: "vector3"
      input:in2 (1)
        interfacename: "noise_scale_2"
        type: "float"
    multiply:scale_xyz (3)
      type: "float"
      input:in1 (1)
        nodename: "add_xyz"
        type: "float"
      input:in2 (1)
        interfacename: "noise_scale_1"
        type: "float"
    output:out (1)
      nodename: "color_mix"
      type: "color3"
    position:obj_pos (1)
      type: "vector3"
    power:power (3)
      type: "float"
      input:in1 (1)
        nodename: "bias"
        type: "float"
      input:in2 (1)
        interfacename: "noise_power"
        type: "float"
    sin:sin (2)
      type: "float"
      input:in (1)
        nodename: "sum"
        type: "float"
  standard_surface:SR_marble1 (6)
    type: "surfaceshader"
    xpos: "13.768116"
    ypos: "-0.672414"
    input:base (1)
      type: "float"
      value: "1"
    input:base_color (1)
      nodegraph: "NG_marble1"
      output: "out"
      type: "color3"
    input:specular_roughness (1)
      type: "float"
      value: "0.1"
    input:subsurface (1)
      type: "float"
```



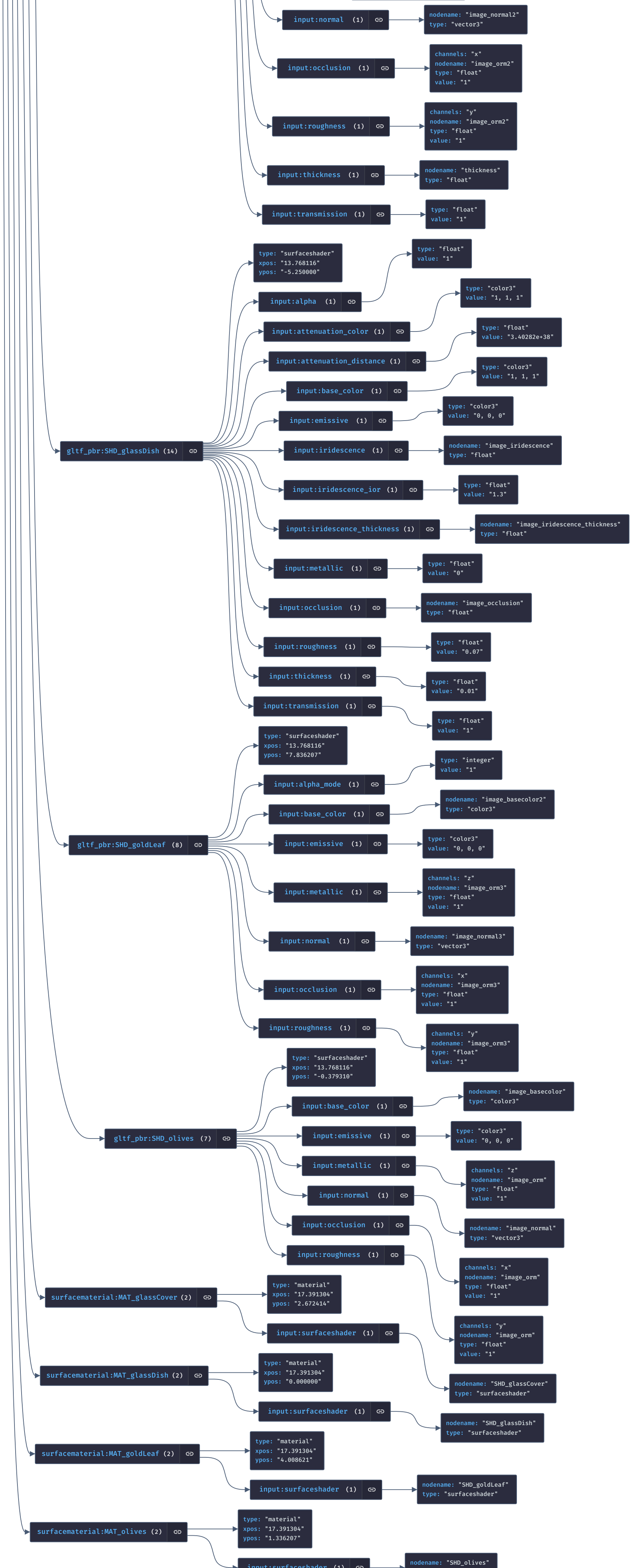
▼ Graph of "Boombox" Example



▼ Graph of "Olives" Example



```
mimetype: "application/mtlx+json"

materialx (23)

version: "1.38"

gltf_image:image_basecolor (2)
    type: "color3"
    xpos: "10.144928"
    ypos: "-1.137931"
    input:file (1)
        colorspace: "srgb_texture"
        type: "filename"
        value: "olives_col.png"

gltf_image:image_basecolor2 (2)
    type: "color3"
    xpos: "10.144928"
    ypos: "10.137931"
    input:file (1)
        colorspace: "srgb_texture"
        type: "filename"
        value: "goldleaf_col.png"

gltf_image:image_iridescence (2)
    type: "float"
    xpos: "10.144928"
    ypos: "-4.396552"
    input:file (1)
        type: "filename"
        value: "glassdish_irid.png"

gltf_image:image_iridescence2 (2)
    type: "float"
    xpos: "10.144928"
    ypos: "5.543103"
    input:file (1)
        type: "filename"
        value: "glasscover_irid.png"

gltf_image:image_occlusion (2)
    type: "float"
    xpos: "10.144928"
    ypos: "-5.732759"
    input:file (1)
        type: "filename"
        value: "goldleaf_orm.png"

gltf_image:image_orm (2)
    type: "vector3"
    xpos: "10.144928"
    ypos: "0.198276"
    input:file (1)
        type: "filename"
        value: "olives_orm.png"

gltf_image:image_orm2 (2)
    type: "vector3"
    xpos: "10.144928"
    ypos: "2.870690"
    input:file (1)
        type: "filename"
        value: "glasscover_orm.png"

gltf_image:image_orm3 (2)
    type: "vector3"
    xpos: "10.144928"
    ypos: "11.474138"
    input:file (1)
        type: "filename"
        value: "goldleaf_orm.png"

gltf_image:thickness (2)
    type: "float"
    xpos: "10.144928"
    ypos: "8.801724"
    input:file (1)
        type: "filename"
        value: "glasscover_thick.png"

gltf_iridescence_thickness:image_iridescence_thickne…(4)
    type: "float"
    xpos: "10.144928"
    ypos: "-3.060345"
    input:file (1)
        type: "filename"
        value: "glassdish_irid.png"
    input:thicknessMax (1)
        type: "float"
        value: "550"
    input:thicknessMin (1)
        type: "float"
        value: "500"

gltf_iridescence_thickness:image_iridescence_thicknes…(4)
    type: "float"
    xpos: "10.144928"
    ypos: "6.879310"
    input:file (1)
        type: "filename"
        value: "glasscover_irid.png"
    input:thicknessMax (1)
        type: "float"
        value: "550"
    input:thicknessMin (1)
        type: "float"
        value: "500"

gltf_normalmap:image_normal (2)
    type: "vector3"
    xpos: "10.144928"
    ypos: "1.534483"
    input:file (1)
        type: "filename"
        value: "olives_nrm.png"

gltf_normalmap:image_normal2 (2)
    type: "vector3"
    xpos: "10.144928"
    ypos: "4.206897"
    input:file (1)
        type: "filename"
        value: "glasscover_nrm.png"

gltf_normalmap:image_normal3 (2)
    type: "vector3"
    xpos: "10.144928"
    ypos: "12.810345"
    input:file (1)
        type: "filename"
        value: "goldleaf_nrm.png"

gltf_pbr:SHD_glassCover (16)
    type: "surfaceshader"
    xpos: "13.768116"
    ypos: "2.396552"
    input:alpha (1)
        type: "float"
        value: "1"
    input:attenuation_color (1)
        type: "color3"
        value: "1, 1, 1"
    input:attenuation_distance (1)
        type: "float"
        value: "3.40282e+38"
    input:base_color (1)
        type: "color3"
        value: "1, 1, 1"
    input:emissive (1)
        type: "color3"
        value: "0, 0, 0"
    input:ior (1)
        type: "float"
        value: "1.5"
    input:iridescence (1)
        nodename: "image_iridescence2"
        type: "float"
    input:iridescence_ior (1)
        type: "float"
        value: "1.3"
    input:iridescence_thickness (1)
        nodename: "image_iridescence_thickness2"
        type: "float"
    input:metallic (1)
        channels: "z"
        nodename: "image_orm2"
        type: "float"
        value: "1"
```



input:normal (1)
nodename: "image_normal2"
type: "vector3"

input:occlusion (1)
channels: "x"
nodename: "image_orm2"
type: "float"
value: "1"

input:roughness (1)
channels: "y"
nodename: "image_orm2"
type: "float"
value: "1"

input:thickness (1)
nodename: "thickness"
type: "float"

input:transmission (1)
type: "float"
value: "1"

gltf_pbr:SHD_glassDish (14)
type: "surfaceshader"
xpos: "13.768116"
ypos: "-5.250000"

input:alpha (1)
type: "float"
value: "1"

input:attenuation_color (1)
type: "color3"
value: "1, 1, 1"

input:attenuation_distance (1)
type: "float"
value: "3.40282e+38"

input:base_color (1)
type: "color3"
value: "1, 1, 1"

input:emissive (1)
type: "color3"
value: "0, 0, 0"

input:iridescence (1)
nodename: "image_iridescence"
type: "float"

input:iridescence_ior (1)
type: "float"
value: "1.3"

input:iridescence_thickness (1)
nodename: "image_iridescence_thickness"
type: "float"

input:metallic (1)
type: "float"
value: "0"

input:occlusion (1)
nodename: "image_occlusion"
type: "float"

input:roughness (1)
type: "float"
value: "0.07"

input:thickness (1)
type: "float"
value: "0.01"

input:transmission (1)
type: "float"
value: "1"

gltf_pbr:SHD_goldLeaf (8)
type: "surfaceshader"
xpos: "13.768116"
ypos: "7.836207"

input:alpha_mode (1)
type: "integer"
value: "1"

input:base_color (1)
nodename: "image_basecolor2"
type: "color3"

input:emissive (1)
type: "color3"
value: "0, 0, 0"

input:metallic (1)
channels: "z"
nodename: "image_orm3"
type: "float"
value: "1"

input:normal (1)
nodename: "image_normal3"
type: "vector3"

input:occlusion (1)
channels: "x"
nodename: "image_orm3"
type: "float"
value: "1"

input:roughness (1)
channels: "y"
nodename: "image_orm3"
type: "float"
value: "1"

gltf_pbr:SHD_olives (7)
type: "surfaceshader"
xpos: "13.768116"
ypos: "-0.379310"

input:base_color (1)
nodename: "image_basecolor"
type: "color3"

input:emissive (1)
type: "color3"
value: "0, 0, 0"

input:metallic (1)
channels: "z"
nodename: "image_orm"
type: "float"
value: "1"

input:normal (1)
nodename: "image_normal"
type: "vector3"

input:occlusion (1)
channels: "x"
nodename: "image_orm"
type: "float"
value: "1"

input:roughness (1)
channels: "y"
nodename: "image_orm"
type: "float"
value: "1"

surfacematerial:MAT_glassCover (2)
type: "material"
xpos: "17.391304"
ypos: "2.672414"

input:surfaceshader (1)
nodename: "SHD_glassCover"
type: "surfaceshader"

surfacematerial:MAT_glassDish (2)
type: "material"
xpos: "17.391304"
ypos: "0.000000"

input:surfaceshader (1)
nodename: "SHD_glassDish"
type: "surfaceshader"

surfacematerial:MAT_goldLeaf (2)
type: "material"
xpos: "17.391304"
ypos: "4.008621"

input:surfaceshader (1)
nodename: "SHD_goldLeaf"
type: "surfaceshader"

surfacematerial:MAT_olives (2)
type: "material"
xpos: "17.391304"
ypos: "1.336207"

input:surfaceshader (1)
nodename: "SHD_olives"

Deserialization from JSON

For conversion from JSON we introduce two functions:

1. `documentFromJSON()` : Converts a JSON string to a MaterialX document.
2. `elementFromJSON()` : Converts a JSON string to a MaterialX element, and continues to recursively convert any children.

Note that to perform deserialization we need to split the `category` and `name` out for non-attribute elements.

In [ ]:

```
# Separator between category and name in JSON element
JSON_CATEGORY_NAME_SEPARATOR = ":"

# Convert JSON element to MaterialX
def elementFromJSON(node, elem):
    '''
    Convert an JSON element to MaterialX
    '''
    for key in node:
        value = node[key]

        # Set attributes
        if isinstance(value, str):
            elem.setAttribute(key, str(value))

        # Traverse chilren
        else:
            # Traverse down from root
            if key == MATERIALX_DOCUMENT_ROOT:
                elementFromJSON(value, elem)
                continue

            # Split key name by ":" to get category and name
            category, name = key.split(JSON_CATEGORY_NAME_SEPARATOR, 1)
            if category and not elem.getChild(name):
                child = elem.addChildOfCategory(category, name)
                elementFromJSON(value, child)

# Convert JSON to MaterialX document
def documentFromJSON(jsonDoc, doc):
    '''
    Convert a JSON document to MaterialX
    '''
    elementFromJSON(jsonDoc, doc)
```

Using these functions we can convert the JSON document and NodeGraph results back to a MaterialX documents.

We also add in validation and an "upgrade" call to roughly match what is performed for XML serialization.

In [ ]:

```
# Convert entire document back from JSON
newDoc = mx.createDocument()
jsonObject = json.loads(doc_result)
documentFromJSON(jsonObject, newDoc)

# Validate and upgrade element version
```

```python
valid, errors = newDoc.validate()
if not valid:
    print('Validation errors:')
    for err in errors:
        print('  {}'.format(err))
newDoc.upgradeVersion()

newDocString = mx.writeToXmlString(newDoc)
text = '<details><summary>JSON Deserialization of Document</summary>\n\n' + '```xml\n' + newDocS
display_markdown(text, raw=True)
```

▼ JSON Deserialization of Document

```
<?xml version="1.0"?>
<materialx version="1.38" colorspace="lin_rec709" fileprefix="../../../Images/">
  <nodegraph name="NG_BrickPattern">
    <input name="brick_color" type="color3" value="0.661876, 0.19088, 0" uiname="Brick Color" uifolder="Color" />
    <input name="hue_variation" type="float" value="0.083" uimin="0" uimax="1" uiname="Hue Variation" uifolder="Color" />
    <input name="value_variation" type="float" value="0.787" uimin="0" uimax="1" uiname="Value Variation" uifolder="Color" />
    <input name="roughness_amount" type="float" value="0.853" uimin="0" uimax="1" uiname="Roughness Amount" uifolder="Roughness" />
    <input name="dirt_color" type="color3" value="0.56372, 0.56372, 0.56372" uiname="Dirt Color" uifolder="Dirt" />
    <input name="dirt_amount" type="float" value="0.248" uimin="0" uimax="1" uiname="Dirt Amount" uifolder="Dirt" />
    <input name="uvtiling" type="float" value="3" uisoftmin="1" uisoftmax="16" uiname="UV Tiling" uifolder="Texturing" />
    <multiply name="node_multiply_5" type="color3">
      <input name="in1" type="color3" nodename="node_mix_6" />
      <input name="in2" type="float" nodename="node_tiledimage_float_7" />
    </multiply>
    <mix name="node_mix_8" type="color3">
      <input name="fg" type="color3" nodename="node_multiply_5" />
      <input name="bg" type="color3" nodename="node_multiply_9" />
      <input name="mix" type="float" nodename="node_tiledimage_float_10" />
    </mix>
    <constant name="node_color_11" type="color3">
      <input name="value" type="color3" value="0.263273, 0.263273, 0.263273" />
    </constant>
    <multiply name="node_multiply_9" type="color3">
      <input name="in1" type="color3" nodename="node_color_11" />
      <input name="in2" type="float" nodename="node_tiledimage_float_7" />
    </multiply>
    <rgbtohsv name="node_rgbtohsv_12" type="color3">
      <input name="in" type="color3" interfacename="brick_color" />
    </rgbtohsv>
    <combine3 name="node_combine3_color3_13" type="color3">
      <input name="in1" type="float" nodename="node_multiply_14" />
      <input name="in2" type="float" value="0" />
      <input name="in3" type="float" nodename="node_multiply_15" />
    </combine3>
    <add name="node_add_16" type="color3">
      <input name="in1" type="color3" nodename="node_combine3_color3_13" />
      <input name="in2" type="color3" nodename="node_rgbtohsv_12" />
    </add>
    <hsvtorgb name="node_hsvtorgb_17" type="color3">
  <input name="in" type="color3" nodename="node_add_16" />
</hsvtorgb>
<subtract name="node_subtract_18" type="float">
  <input name="in1" type="float" nodename="node_add_19" />
  <input name="in2" type="float" value="0.35" />
</subtract>
<multiply name="node_multiply_14" type="float">
  <input name="in1" type="float" nodename="node_subtract_18" />
  <input name="in2" type="float" interfacename="hue_variation" />
</multiply>
<multiply name="node_multiply_15" type="float">
  <input name="in1" type="float" nodename="node_add_19" />
  <input name="in2" type="float" nodename="node_multiply_20" />
</multiply>
<clamp name="node_clamp_0" type="color3">
  <input name="in" type="color3" nodename="node_mix_8" />
</clamp>
<multiply name="node_multiply_1" type="float">
  <input name="in1" type="float" nodename="node_divide_21" />
  <input name="in2" type="float" nodename="node_tiledimage_float_22" />
</multiply>
<max name="node_max_1" type="float">
  <input name="in1" type="float" nodename="node_tiledimage_float_10" />
  <input name="in2" type="float" value="0.00001" />
</max>
<divide name="node_divide_21" type="float">
  <input name="in1" type="float" interfacename="roughness_amount" />
  <input name="in2" type="float" nodename="node_max_1" />
</divide>
<mix name="node_mix_6" type="color3">
  <input name="fg" type="color3" interfacename="dirt_color" />
  <input name="bg" type="color3" nodename="node_hsvtorgb_17" />
  <input name="mix" type="float" nodename="node_multiply_23" />
</mix>
<multiply name="node_multiply_23" type="float">
  <input name="in1" type="float" interfacename="dirt_amount" />
  <input name="in2" type="float" nodename="node_tiledimage_float_24" />
</multiply>
<multiply name="node_multiply_25" type="float">
  <input name="in1" type="float" interfacename="hue_variation" />
  <input name="in2" type="float" nodename="node_tiledimage_float_26" />
</multiply>
<add name="node_add_19" type="float">
  <input name="in1" type="float" nodename="node_multiply_25" />
  <input name="in2" type="float" nodename="node_tiledimage_float_7" />
</add>
<multiply name="node_multiply_20" type="float">
```

```
      <input name="in1" type="float" interfacename="value_variation" />
      <input name="in2" type="float" nodename="node_tiledimage_float_26" />
    </multiply>
    <normalmap name="node_normalmap_3" type="vector3">
      <input name="in" type="vector3" nodename="node_tiledimage_vector3_27" />
    </normalmap>
    <convert name="node_convert_1" type="vector2">
      <input name="in" type="float" interfacename="uvtiling" />
    </convert>
    <tiledimage name="node_tiledimage_vector3_27" type="vector3">
      <input name="file" type="filename" value="brick_normal.jpg" />
      <input name="uvtiling" type="vector2" nodename="node_convert_1" />
    </tiledimage>
    <tiledimage name="node_tiledimage_float_22" type="float">
      <input name="file" type="filename" value="brick_roughness.jpg" />
      <input name="uvtiling" type="vector2" nodename="node_convert_1" />
    </tiledimage>
    <tiledimage name="node_tiledimage_float_10" type="float">
      <input name="file" type="filename" value="brick_mask.jpg" />
      <input name="uvtiling" type="vector2" nodename="node_convert_1" />
    </tiledimage>
    <tiledimage name="node_tiledimage_float_7" type="float">
      <input name="file" type="filename" value="brick_base_gray.jpg" />
      <input name="uvtiling" type="vector2" nodename="node_convert_1" />
    </tiledimage>
    <tiledimage name="node_tiledimage_float_26" type="float">
      <input name="file" type="filename" value="brick_variation_mask.jpg" />
      <input name="uvtiling" type="vector2" nodename="node_convert_1" />
    </tiledimage>
    <tiledimage name="node_tiledimage_float_24" type="float">
      <input name="file" type="filename" value="brick_dirt_mask.jpg" />
      <input name="uvtiling" type="vector2" nodename="node_convert_1" />
    </tiledimage>
    <output name="base_color_output" type="color3" nodename="node_clamp_0" />
    <output name="specular_roughness_output" type="float" nodename="node_multiply_1" />
    <output name="normal_output" type="vector3" nodename="node_normalmap_3" />
  </nodegraph>
  <standard_surface name="N_StandardSurface" type="surfaceshader">
    <input name="base_color" type="color3" nodegraph="NG_BrickPattern" output="base_color_output" />
    <input name="specular_roughness" type="float" nodegraph="NG_BrickPattern" output="specular_roughness_output" />
    <input name="normal" type="vector3" nodegraph="NG_BrickPattern" output="normal_output" />
  </standard_surface>
  <surfacematerial name="M_BrickPattern" type="material">
    <input name="surfaceshader" type="surfaceshader" nodename="N_StandardSurface" />
  </surfacematerial>
</materialx>
```

In [ ]:

```python
# Convert nodegraph interface back from JSON
newDoc = mx.createDocument()
jsonObject = json.loads(graph_result)
documentFromJSON(jsonObject, newDoc)

# Validate and upgrade element version
valid, errors = newDoc.validate()
if not valid:
    print('Validation errors:')
    for err in errors:
        print('  {}'.format(err))
newDoc.upgradeVersion()

newDocString = mx.writeToXmlString(newDoc)
text = '<details><summary>JSON Deserialization of NodeGraph</summary>\n\n' + '```xml\n' + newDocS
display_markdown(text, raw=True)
```

▼ JSON Deserialization of NodeGraph

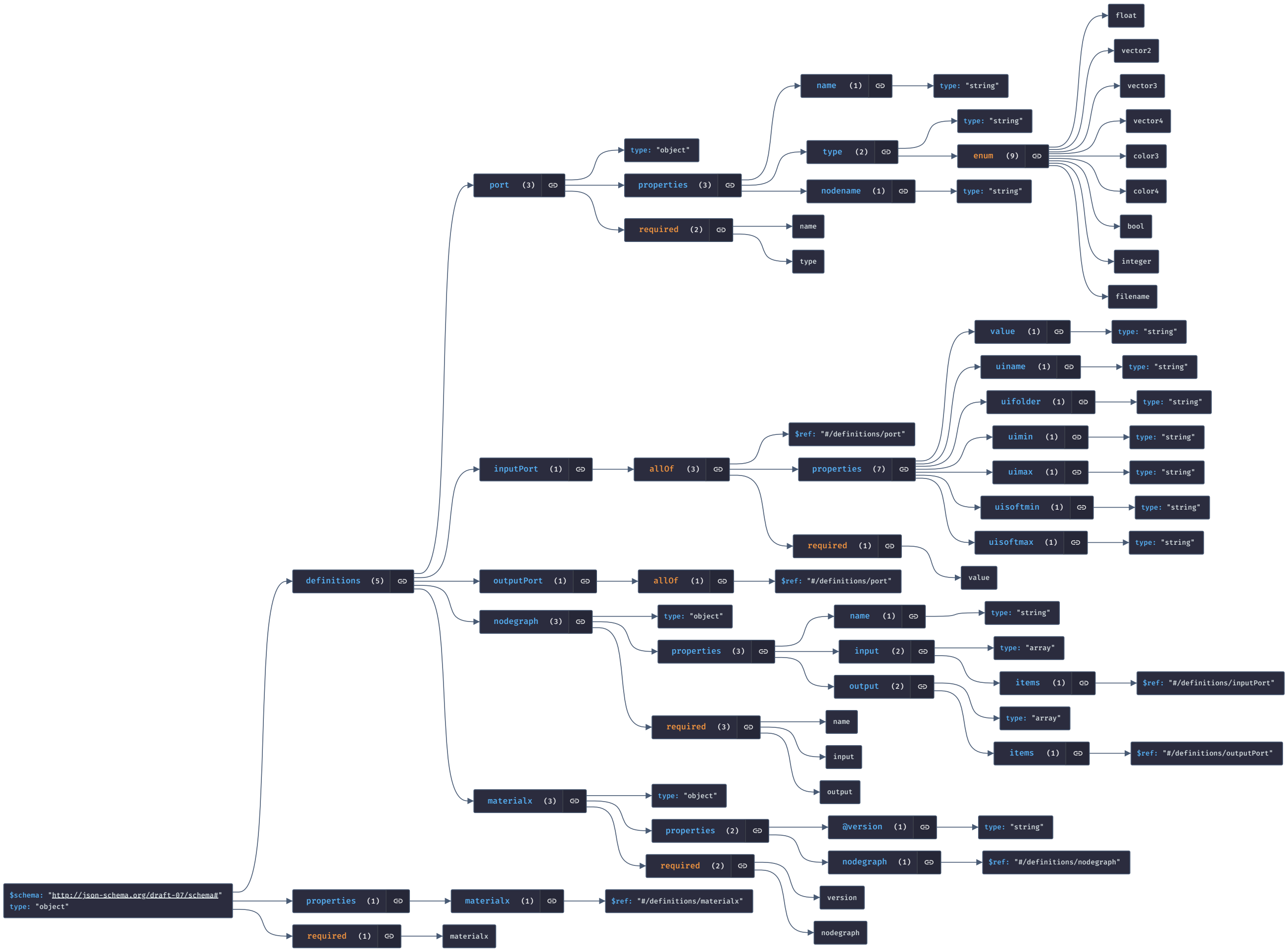
```xml
<?xml version="1.0"?>
<materialx version="1.38">
  <nodegraph name="NG_BrickPattern">
    <input name="brick_color" type="color3" value="0.661876, 0.19088, 0" uiname="Brick Color" uifolder="Color" />
    <input name="hue_variation" type="float" value="0.083" uimin="0" uimax="1" uiname="Hue Variation" uifolder="Color" />
    <input name="value_variation" type="float" value="0.787" uimin="0" uimax="1" uiname="Value Variation" uifolder="Color" />
    <input name="roughness_amount" type="float" value="0.853" uimin="0" uimax="1" uiname="Roughness Amount" uifolder="Roughness" />
    <input name="dirt_color" type="color3" value="0.56372, 0.56372, 0.56372" uiname="Dirt Color" uifolder="Dirt" />
    <input name="dirt_amount" type="float" value="0.248" uimin="0" uimax="1" uiname="Dirt Amount" uifolder="Dirt" />
    <input name="uvtiling" type="float" value="3" uisoftmin="1" uisoftmax="16" uiname="UV Tiling" uifolder="Texturing" />
    <output name="base_color_output" type="color3" />
    <output name="specular_roughness_output" type="float" />
    <output name="normal_output" type="vector3" />
  </nodegraph>
</materialx>
```

Obtaining a JSON Schema for MaterialX

For this example we will use the first test results just the `NodeGraph`` for simplicity. It is possible to create a schema for all elements of MaterialX which would include all the definitions which are part of the standard library.

The schema is created using the OpenAI Python package. Various MaterialX documents (saved out in JSON) were used as input data.

A graph of the schema is shown below, with the textual description below:



Some examples and test suite documents are suitable to obtain most of the schema. Small edits we're made for anything amiss such setting "required" attributes.

Note that this is just a sample schema. It is not complete. It is also not the only possible schema.

In [ ]:
```
# Define the JSON schema.
schema = {
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "port": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "type": {
          "type": "string",
          "enum": ["float", "vector2", "vector3", "vector4", "color3", "color4", "bool", "intege
        },
        "nodename": {
          "type": "string"
        }
      },
      "required": ["name", "type"]
    },
    "inputPort": {
      "allOf": [
        {
          "$ref": "#/definitions/port"
        },
        {
          "properties": {
            "value": {
               "type" : "string"
            },
            "uiname": {
              "type": "string"
            },
            "uifolder": {
              "type": "string"
            },
            "uimin": {
              "type": "string"
            },
            "uimax": {
              "type": "string"
            },
            "uisoftmin": {
              "type": "string"
            },
            "uisoftmax": {
              "type": "string"
            }
          },
          "required": ["value"]
        }
      ]
    },
    "outputPort": {
      "allOf": [
        {
          "$ref": "#/definitions/port"
        }
      ]
    },
    "nodegraph": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "input": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/inputPort"
          }
        },
        "output": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/outputPort"
          }
        }
      },
      "required": ["name", "input", "output"]
    },
    "materialx": {
      "type": "object",
      "properties": {
        "@version": {
          "type": "string"
        },
        "nodegraph": {
          "$ref": "#/definitions/nodegraph"
        }
      },
      "required": ["version", "nodegraph"]
    }
  },
  "type": "object",
  "properties": {
    "materialx": {
      "$ref": "#/definitions/materialx"
    }
  },
  "required": ["materialx"]
}

# Write the schema to a file
jsonSchemaFilePath = 'data/mtlx_sample_schema.json'
with open(jsonSchemaFilePath, 'w') as f:
    json.dump(schema, f)

# Read the schema from a file
loaded_schema = {}
with open(jsonSchemaFilePath, 'r') as schema_file:
    loaded_schema = json.loads(schema_file.read())

text = '<details><summary>MaterialX JSON Schema</summary>\n\n' + '```json\n' + json.dumps(schema
display_markdown(text, raw=True)
```

▼ MaterialX JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "port": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "type": {
          "type": "string",
          "enum": [
            "float",
            "vector2",
            "vector3",
            "vector4",
            "color3",
            "color4",
            "bool",
            "integer",
            "filename"
          ]
        },
        "nodename": {
          "type": "string"
        }
      },
      "required": [
        "name",
        "type"
      ]
    },
    "inputPort": {
      "allOf": [
        {
          "$ref": "#/definitions/port"
        },
        {
          "properties": {
            "value": {
              "type": "string"
            },
            "uiname": {
              "type": "string"
            },
            "uifolder": {
```

```
          "type": "string"
        },
        "uimin": {
          "type": "string"
        },
        "uimax": {
          "type": "string"
        },
        "uisoftmin": {
          "type": "string"
        },
        "uisoftmax": {
          "type": "string"
        }
      },
      "required": [
        "value"
      ]
    }
  ]
},
"outputPort": {
  "allOf": [
    {
      "$ref": "#/definitions/port"
    }
  ]
},
"nodegraph": {
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "input": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/inputPort"
      }
    },
    "output": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/outputPort"
      }
    }
  },
      "required": [
        "name",
        "input",
        "output"
      ]
    },
    "materialx": {
      "type": "object",
      "properties": {
        "@version": {
          "type": "string"
        },
        "nodegraph": {
          "$ref": "#/definitions/nodegraph"
        }
      },
      "required": [
        "version",
        "nodegraph"
      ]
    }
  },
  "type": "object",
  "properties": {
    "materialx": {
      "$ref": "#/definitions/materialx"
    }
  },
  "required": [
    "materialx"
  ]
}
```

Validating MaterialX JSON with Schema

To perform validation we will use the `validate` interface from the `jsonschema` package.

In [ ]:

```
# Validate JSON formatted MaterialX document.
from jsonschema import validate as jsvalidate

# Validate the JSON string against the schema
def validateJson(json_string, schema):
    try:
        data = json.loads(json_string)
        jsvalidate(data, schema)
    except Exception as e:
        return e
    return ''
```

```
error = validateJson(json_string, loaded_schema)
if not error:
  print("JSON string is valid against the schema")
else:
  print('JSON string is not valid:\n')
  print(error)
```

```
JSON string is valid against the schema
```