# 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

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

▼ Extracted MaterialX Nodegraphs in 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>
```

# Conversion of Graphs 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 [ ]:
```
# Build JSON from XML
python_dict = xmltodict.parse(xml_string)
json_string = json.dumps(python_dict)
json_string_fmt = json.dumps(python_dict, indent=2)

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

▼ 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"
      }
    ]
  }
 }
}
```


## Obtaining a JSON Schema for MaterialX

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

The standard library as well as 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.

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 open><summary>MaterialX JSON Schema</summary>\n\n' + '```json\n' + json.dumps(s
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 `jsconschema` package.

In [ ]:
```python
# 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
```