In [36]:
```
import MaterialX as mx

print("MaterialX version:", mx.getVersionString())

def createTestDoc():
    doc : mx.Document = mx.createDocument()

    stdlib = mx.createDocument()
    libFiles = mx.loadLibraries(mx.getDefaultDataLibraryFolders(), mx.getDefaultDat

    defs = stdlib.getNodeDefs()
    print("Number of node definitions loaded:", len(defs))
    return doc, stdlib

doc, stdlib = createTestDoc()
```

```
MaterialX version: 1.39.5
Number of node definitions loaded: 803
```

In [37]:
```
def test_make_functional_definition(test_name):
    test_def = stdlib.getNodeDef(test_name)
    if test_def:
        print(f"Node Definition '{test_name}' found.")
        #print(mx.prettyPrint(test_def))

        graph = test_def.getImplementation()
        if graph:
            if graph.isA(mx.NodeGraph) :
                #print(mx.prettyPrint(graph))
                newGraph = test_def.makeFunctionalDefinition()

                if False and not newGraph:
                    # This is the C++ code in Python form
                    ngname = graph.getNodeDefString()
                    qualname = graph.getQualifiedName(test_def.getName())
                    print("nodedef string:", ngname)
                    print("qualified name:", qualname)
                    if ngname == qualname:
                        newGraphName = graph.getName()
                        newGraph = test_def.addChildOfCategory("nodegraph", newGrap
                        if not newGraph:
                            print("Failed to create new functional node graph:", ne
                        else:
                            print("Created new functional node graph:", newGraphNam
                            newGraph.copyContentFrom(graph)
                            newGraph.removeAttribute(mx.InterfaceElement.NODE_DEF_A

                            parent = test_def.getParent()
                            graph.removeAttribute(mx.InterfaceElement.NODE_DEF_ATTR
                            #tempName = parent.createValidChildName(newGraphName +
                            #graph.setName(tempName)


                if newGraph:
                    print("New functional node definition name:", newGraph.getName(
                    print(mx.prettyPrint(test_def))

test_name = "ND_tiledimage_color3"
test_make_functional_definition(test_name)

test_name = "ND_tiledimage_color4"
test_make_functional_definition(test_name)
```

```
Node Definition 'ND_tiledimage_color3' found.
New functional node definition name: NG_tiledimage_color3
<nodedef name="ND_tiledimage_color3" node="tiledimage" nodegroup="texture2d">
  <input name="file" type="filename" value="" uniform="true">
  <input name="default" type="color3" value="0.0, 0.0, 0.0">
  <input name="texcoord" type="vector2" defaultgeomprop="UV0">
  <input name="uvtiling" type="vector2" value="1.0, 1.0">
  <input name="uvoffset" type="vector2" value="0.0, 0.0">
  <input name="realworldimagesize" type="vector2" value="1.0, 1.0" unittype="distanc
e">
  <input name="realworldtilesize" type="vector2" value="1.0, 1.0" unittype="distanc
e">
  <input name="filtertype" type="string" value="linear" enum="closest,linear,cubic"
uniform="true">
  <input name="framerange" type="string" value="" uniform="true">
  <input name="frameoffset" type="integer" value="0" uniform="true">
  <input name="frameendaction" type="string" value="constant" enum="constant,clamp,p
eriodic,mirror" uniform="true">
  <output name="out" type="color3" default="0.0, 0.0, 0.0">
  <nodegraph name="NG_tiledimage_color3">
    <multiply name="N_mult_color3" type="vector2">
      <input name="in1" type="vector2" interfacename="texcoord">
      <input name="in2" type="vector2" interfacename="uvtiling">
    <subtract name="N_sub_color3" type="vector2">
      <input name="in1" type="vector2" nodename="N_mult_color3">
      <input name="in2" type="vector2" interfacename="uvoffset">
    <divide name="N_divtilesize_color3" type="vector2">
      <input name="in1" type="vector2" nodename="N_sub_color3">
      <input name="in2" type="vector2" interfacename="realworldimagesize">
    <multiply name="N_multtilesize_color3" type="vector2">
      <input name="in1" type="vector2" nodename="N_divtilesize_color3">
      <input name="in2" type="vector2" interfacename="realworldtilesize">
    <image name="N_img_color3" type="color3">
      <input name="file" type="filename" interfacename="file">
      <input name="default" type="color3" interfacename="default">
      <input name="texcoord" type="vector2" nodename="N_multtilesize_color3">
      <input name="uaddressmode" type="string" value="periodic">
      <input name="vaddressmode" type="string" value="periodic">
      <input name="filtertype" type="string" interfacename="filtertype">
      <input name="framerange" type="string" interfacename="framerange">
      <input name="frameoffset" type="integer" interfacename="frameoffset">
      <input name="frameendaction" type="string" interfacename="frameendaction">
    <output name="out" type="color3" nodename="N_img_color3">

Node Definition 'ND_tiledimage_color4' found.
New functional node definition name: NG_tiledimage_color4
<nodedef name="ND_tiledimage_color4" node="tiledimage" nodegroup="texture2d">
  <input name="file" type="filename" value="" uniform="true">
  <input name="default" type="color4" value="0.0, 0.0, 0.0, 0.0">
  <input name="texcoord" type="vector2" defaultgeomprop="UV0">
  <input name="uvtiling" type="vector2" value="1.0, 1.0">
  <input name="uvoffset" type="vector2" value="0.0, 0.0">
  <input name="realworldimagesize" type="vector2" value="1.0, 1.0" unittype="distanc
e">
  <input name="realworldtilesize" type="vector2" value="1.0, 1.0" unittype="distanc
e">
```

```
  <input name="filtertype" type="string" value="linear" enum="closest,linear,cubic"
uniform="true">
  <input name="framerange" type="string" value="" uniform="true">
  <input name="frameoffset" type="integer" value="0" uniform="true">
  <input name="frameendaction" type="string" value="constant" enum="constant,clamp,p
eriodic,mirror" uniform="true">
  <output name="out" type="color4" default="0.0, 0.0, 0.0, 0.0">
  <nodegraph name="NG_tiledimage_color4">
    <multiply name="N_mult_color4" type="vector2">
      <input name="in1" type="vector2" interfacename="texcoord">
      <input name="in2" type="vector2" interfacename="uvtiling">
    <subtract name="N_sub_color4" type="vector2">
      <input name="in1" type="vector2" nodename="N_mult_color4">
      <input name="in2" type="vector2" interfacename="uvoffset">
    <divide name="N_divtilesize_color4" type="vector2">
      <input name="in1" type="vector2" nodename="N_sub_color4">
      <input name="in2" type="vector2" interfacename="realworldimagesize">
    <multiply name="N_multtilesize_color4" type="vector2">
      <input name="in1" type="vector2" nodename="N_divtilesize_color4">
      <input name="in2" type="vector2" interfacename="realworldtilesize">
    <image name="N_img_color4" type="color4">
      <input name="file" type="filename" interfacename="file">
      <input name="default" type="color4" interfacename="default">
      <input name="texcoord" type="vector2" nodename="N_multtilesize_color4">
      <input name="uaddressmode" type="string" value="periodic">
      <input name="vaddressmode" type="string" value="periodic">
      <input name="filtertype" type="string" interfacename="filtertype">
      <input name="framerange" type="string" interfacename="framerange">
      <input name="frameoffset" type="integer" interfacename="frameoffset">
      <input name="frameendaction" type="string" interfacename="frameendaction">
    <output name="out" type="color4" nodename="N_img_color4">
```

In [38]:
```python
def get_matching_definitions(def_name):
    stdsurf = stdlib.getNodeDef(def_name)
    nodegraph_counts = {}  # Will store {nodegraph: count}
    nodegraph_nodedefs = {}  # Will store {nodegraph: set(nodedefs)}
    if stdsurf:
        print(f"got node def: {stdsurf.getVersionString()}")

        other_stdsurf = stdsurf.getMatchingDefinitions()
        print("* number of matching definitions:", len(other_stdsurf))
        for ndstring in other_stdsurf:
            print("matching definition:", ndstring)
            nd = stdlib.getNodeDef(ndstring)
            if nd:
                mapped_other_stdsurf = stdlib.getMatchingIndirectImplementations(nd
                print("number of mapped implementations:", len(mapped_other_stdsurf
                for impl in mapped_other_stdsurf:
                    print("- mapping implementations:", impl.getName())

                print("  version:", nd.getVersionString(), " inherits from:", nd.ge
                impl = nd.getImplementation()
                if impl.isA(mx.NodeGraph):
                    print("  nodegraph implementation:", impl.getName())
                else:
                    nodegraph_name = None
                    nodegraph_string = impl.getAttribute("nodegraph")
                    if nodegraph_string:
                        impl = stdlib.getNodeGraph(nodegraph_string)
                        nodegraph_name = nodegraph_string
                    if nodegraph_name:
                        # Count usage of each nodegraph
                        if nodegraph_name in nodegraph_counts:
                            nodegraph_counts[nodegraph_name] += 1
                        else:
                            nodegraph_counts[nodegraph_name] = 1
                        # Track which nodedefs use this nodegraph
                        if nodegraph_name not in nodegraph_nodedefs:
                            nodegraph_nodedefs[nodegraph_name] = set()
                        nodegraph_nodedefs[nodegraph_name].add(ndstring)
                    if impl and impl.isA(mx.NodeGraph):
                        print("  mapped implementation:", impl.getName())

        impls = stdlib.getImplementations()
        print("*"*80)
        print("Mapped Node graph usage counts:")
        for ngname, count in nodegraph_counts.items():
            ndefs = nodegraph_nodedefs[ngname]
            for ndef in ndefs:
                print(f"  nodedef: {ndef}, nodegraph: {ngname}, count: {count}")

        print("-"*80 + "\n")


get_matching_definitions("ND_UsdUVTexture")
get_matching_definitions("ND_standard_surface_surfaceshader")
```

```
got node def: 2.2
* number of matching definitions: 2
matching definition: ND_UsdUVTexture
number of mapped implementations: 0
  version: 2.2  inherits from: ND_UsdUVTexture_23
  nodegraph implementation: IMP_UsdUVTexture_22
matching definition: ND_UsdUVTexture_23
number of mapped implementations: 0
  version: 2.3  inherits from:
  nodegraph implementation: IMP_UsdUVTexture_23
********************************************************************************
Mapped Node graph usage counts:
--------------------------------------------------------------------------------

got node def: 1.0.1
* number of matching definitions: 2
matching definition: ND_standard_surface_surfaceshader
number of mapped implementations: 2
- mapping implementations: IMPL_standard_surface_surfaceshader_101
- mapping implementations: IMPL_standard_surface_surfaceshader_optim
  version: 1.0.1  inherits from: ND_standard_surface_surfaceshader_100
  nodegraph implementation: NG_standard_surface_surfaceshader_100
matching definition: ND_standard_surface_surfaceshader_100
number of mapped implementations: 1
- mapping implementations: IMPL_standard_surface_surfaceshader_100
  version: 1.0.0  inherits from:
  nodegraph implementation: NG_standard_surface_surfaceshader_100
********************************************************************************
Mapped Node graph usage counts:
--------------------------------------------------------------------------------
```

In [39]:

```
indirect_mapped_nodedefs = {}
direct_mapped_nodefs = {}

for ndef in stdlib.getNodeDefs():
    impl = ndef.getImplementation()
    if impl:
        if impl.isA(mx.NodeGraph) :
            direct_mapped_nodefs[ndef.getName()] = impl
            continue
        elif impl.hasAttribute("nodegraph"):
            indirect_mapped_nodedefs[ndef.getName()] = impl

print("Directly mapped nodedefs count:", len(direct_mapped_nodefs))
print("Indirectly mapped nodedefs count:", len(indirect_mapped_nodedefs))
if len(indirect_mapped_nodedefs) > 0:
    print("Indirectly mapped nodedefs:")
    for ndef_name, impl in indirect_mapped_nodedefs.items():
        print(f"- nodedef: {ndef_name} -> implementation: {impl.getNamePath()}")

def getUnappedImplementation(find_nodedef_name, indirect_mapped_nodedefs):
    if find_nodedef_name in indirect_mapped_nodedefs:
        return indirect_mapped_nodedefs[find_nodedef_name]
    return None
```

```python
impls = stdlib.getImplementations()
for ndef in stdlib.getNodeDefs():
    ndef_name = ndef.getName()
    unmapped_impl = getUnappedImplementation(ndef_name, indirect_mapped_nodedefs)
    if unmapped_impl:
        print(f"- found unmapped implementation {unmapped_impl.getNamePath()} for n
```

```
Directly mapped nodedefs count: 266
Indirectly mapped nodedefs count: 0
```