Rule Files

A rule file specifies one or more rules used to configure the path-building system.

Below is the DTD for a rule file:

```
<!ELEMENT RULES (RULE)*>
<!ELEMENT RULE (PREDICATE,ROUTE)>
<!ELEMENT PREDICATE EMPTY>
<!ATTLIST PREDICATE value CDATA #REQUIRED>
<!ELEMENT ROUTE (STEP)+>
<!ELEMENT STEP ((BEAD,EDGE,SEED?)|
                (BEAD,SEED?,EDGE)|
                (SEED?,BEAD,EDGE)|
                (SEED?,EDGE,BEAD)|
                (EDGE,SEED?,BEAD)|
                (EDGE,BEAD,SEED?)|
                (SEED)|
                (LOOPBACK))>
<!ELEMENT BEAD EMPTY>
<!ATTLIST BEAD name CDATA #REQUIRED>
<!ELEMENT EDGE EMPTY>
<!ATTLIST EDGE name CDATA #REQUIRED>
<!ELEMENT SEED EMPTY>
<!ATTLIST SEED value CDATA #REQUIRED>
<!ELEMENT LOOPBACK EMPTY>
<!ATTLIST LOOPBACK edge CDATA #REQUIRED>
```

Below is a sample rule file containing a single rule with one of each type of step:

```
<RULES>
<RULE>
<PREDICATE value="namespace:"/>
<ROUTE>
<STEP>
<SEED value="namespace:seed=string:foo"/>
<BEAD name="pmtestbead"/>
<EDGE name="encode"/>
</STEP>
<STEP>
<SEED value="namespace:seed=string:foo"/>
</STEP>
<STEP>
<BEAD name="pmtestbead"/>
<EDGE name="encode"/>
</STEP>
<STEP>
<LOOPBACK edge="decode"/>
</STEP>
</ROUTE>
</RULE>
</RULES>
```

Rules are the primary mechanism for configuring Strings. A rule is defined as a sequence of one or more steps to execute when a specific set of conditions are true. The set of conditions is know as the predicate. The steps are known as the route. A predicate is implemented as a registry object that implements the compare interface. The result of the comparison determines whether Strings will execute the route. A route is composed of steps. There are several types of steps described in more detail below.

Step Types

There are four types of STEP element described below.

Edge Steps

Edge steps are the simplest type of step. Edge steps have the following format:

```
<STEP>
<BEAD name="mybead"/>
<EDGE name="encode"/>
</STEP>
```

An edge step explicitly identifies an edge. This will cause the path-building system to route the path to the specified edge. The bead name must match the name of a registered bead schema. The edge name must match an edge name within that schema which is also the name used to register the edge with SOS_Edge_Register.

Seed Steps

Seed steps have the following format:

```
<STEP>
<SEED value="namespace:seed=string:foo"/>
</STEP>
```

Seed steps specify a namespace object to stack on the path. The effect of this is to make all of the names objects in the seed namespace visible in the path, and to hide corresponding objects that were inserted into the path upstream. While in this SDK you will typically see the Strings-provided namespace class used to define seed objects, any class that supports the namespace, stack, and copy interfaces can be used.

Seed-Edge Steps

Seed-Edge steps are just an edge step and a seed step combined into a single step. Seed-Edge steps have the following format:

```
<STEP>
<SEED value="namespace:seed=string:foo"/>
<BEAD name="pmtestbead"/>
<EDGE name="encode"/>
</STEP>
```

The seed part of the step is applied to the path before the edge is inserted into the path.

Loopback Steps

Loopback steps are the most dynamic and powerful type of step. Loopback steps have the following

```
<STEP>
<LOOPBACK edge="decode"/>
</STEP>
```

The LOOPBACK element specifies the name of the edge to use as a source of loopback information. The value of the edge attribute must be the name of one of the edges of the bead currently on top of the loopback stack, and that edge must have a loopback attribute specified in its schema.