# Including Render Information to SBML Layouts
# Version 0.1

Ralph Gauges, Ursula Rost, Sven Sahle and Katja Wegner
European Media Laboratory
Schloss-Wolfsbrunnen Weg 33
69118 Heidelberg
Germany

September 18, 2003

# Introduction

This document is meant to complement the SBML Layout extension document presented earlier (http://projects.villa-bosch.de/bcb/sbml/) with render information for the layout objects. The ideas the went into this render extension are mostly the same as described in the layout extension. We wanted this extension to be as flexible as possible. In oder for the user to make maximal use of defined render objects across several layouts, we choose to separate the render information completely from the layout information in the sbml file. Since this is only a first draft all the tag names should be though of as preliminary.

# Namespace

For the render extensions we use the same namespace as for the layout information since we assume that this document will be merged with the layout extension once it has matured a little. It takes the following form xmlns:sl2="http://projects.eml.org/bcb/sbml/level2/". A SBML file that would utilize the extension could have the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
   <sbml xmlns:sbml="http://www.sbml.org/sbml/level2" level="2" \\
         version="1"
         xmlns:sl2="http://projects.eml.org/bcb/sbml/level2"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://projects.eml.org/bcb/sbml/level2
         http://projects.eml.org/bcb/sbml/level2/layout2.xsd">
```

# Meta information

Most the render classes below are derived from a class called SBase which was taken from the SBML Level 2 schema specification (http://www.sbml.org/sbml/level2/version1/). This enables programs to store meta information with the render objects. As well most objects do have a unique id of type SId.

```
<xsd:complexType name="SBase" abstract="true">
 <xsd:sequence>
  <xsd:element name="notes" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:any namespace="http://www.w3.org/1999/xhtml"
```

```
            processContents="skip"
            maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="annotation" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:any processContents="skip" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="metaid" type="xsd:ID" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="SId">
 <xsd:restriction base="xsd:string">
  <xsd:pattern value="(_|[a-z]|[A-Z])(_|[a-z]|[A-Z]|[0-9])*"/>
 </xsd:restriction>
</xsd:simpleType>
```

# Overall structure

As stated above, the render information has been completely separated from the layout information in the sbml file. It will have a separate tag called **r**ender. There is just one render object per file which contains the rendering information for all specified layouts. This render objects hold several list. A listOfFilltypes holds a list of possible filltypes for shapes. A listOfLinetypes holds all the different line types. A listOfColors will hold a number of predefinedColorNames which might be easier to use than RGBA values. A listOfShapes holds all the complex and primitive shapes that will eventually be rendered. All shapes are derived from a base class Shape. Shapes will consist of curves, text labels, bitmaps and several forms of primitive shapes like circle, rectangle, box sphere etc. A listOfRenderGroups hold groups of render objects as defined in the listOfShapes. Those render groups can than be referenced by the layout part together with a transformation object that is to be used on the render object. This actually differs a little from the mail, that was sent to the sbml mailing list before. In that mail, we additionally had a listOfCurves and a listOfBitmaps, but after thinking about it some more, we thought that those are actually just a special form of shapes.

```
<xsd:complexType name="ListOfLinetypes">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:sequence>
    <xsd:element name="linetype" type="sl2:Linetype"
                 maxOccurs="unbounded"/>
```

```
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfFilltypes">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:sequence>
    <xsd:element name="filltype" type="sl2:Filltype"
                 maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfShapes">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:sequence>
    <xsd:element name="shape" type="sl2:Shape" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfRenderGroups">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:sequence>
    <xsd:element name="group" type="sl2:RenderGroup" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Render">
  <xsd:sequence>
    <xsd:element name="listOfColors" type="sl2:ListOfColors" minOccurs="0"/>
    <xsd:element name="listOfLinetypes" type="sl2:ListOfLinetypes" minOccurs="0"/>
    <xsd:element name="listOfShapes" type="sl2:ListOfShapes" minOccurs="0"/>
    <xsd:element name="listOfRenderGroups" type="sl2:ListOfRenderGroups" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>
```

# Colors

Colors are given in terms of RGBA values. Since it might be more convenient to specify colors names instead of RGBA values, the user has the option to define some colors and give names to them in oder to be referenced in other objects.

```
<xsd:simpleType name="ColorChannel">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="0"/>
  <xsd:maxInclusive value="255"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="Color">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:attribute name="id" type="sl2:SId"/>
   <xsd:attribute name="red" type="sl2:ColorChannel"/>
   <xsd:attribute name="green" type="sl2:ColorChannel"/>
   <xsd:attribute name="blue" type="sl2:ColorChannel"/>
   <xsd:attribute name="alpha" type="sl2:ColorChannel" use="optional" default="255"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

# Linetypes

The ListOfLinetypes holds zero or more Linetype objects. A linetype objects has an attribute for the thickness of the line and one for the stroke.

```
<xsd:complexType name="StrokeType" abstract="true">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
    ....?????......
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Linetype">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:sequence>
     <xsd:element name="stroke" type="sl2:StrokeTypes"/>
   </xsd:sequence>
   <xsd:attribute name="id" type="sl2:SId"/>
   <xsd:attribute name="lineWidth" type="xsd:double"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Filltype" abstract="true">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:attribute name="id" type="sl2:SId"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SingleColorFill">
```

```
 <xsd:complexContent>
  <xsd:extension base="sl2:Filltype">
   <xsd:attribute name="color" type="sl2:Color"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

# Shapes

This will probably be the most difficult part of the render extension and we hope to get much feedback from other people as to what shapes would be needed for the different applications. Eventually many of the shape definitions will probably borrow from corresponding SVG definitions. For 3D shapes, we might borrow from OpenGL or from an upcoming SVG 3D extension. We also have to give thought as to how 2D shapes are to be treated in a 3D environment and to a lesser extend, we have to come up with a way to handle 3D shapes in 2D programs. As there is probably not going to be a program that will do 3D layouts for quite a while yet, we will concentrate on the 2D shapes at first. As stated above all shapes will be derived from a base type called Shape which extends SBase and has a unique id of type SId. Further on we think that i might be helpful to derive to more classes namely Shape2D and Shape3D which will form the basis for all 2D and 3D shapes to come. The 2D shapes that came to our mind so far are curves, text labels, bitmaps, ellipse, rectangle, triangle. Actually rectangle and triangle might be substituted by a more versatile polyline type which would be based on curve. Although this might be more difficult to implement as far as filling shapes is concerned. The curve type is actually special in that it might not fit into the 2D or 3D category and might be directly based on Shape. Also the text label and bitmap types might be difficult to handle when it comes to transformations, especially scaling. We won't give any definitions of Shapes here until some of the issues mentioned above have been discussed in more detail. We first had a color associated with the Linetype (s.o.) but in order to make better use of the defined line types, we moved the lineColor attribute to shapes (at least in thought). Unfortunately this makes it rather difficult to have something like an alternating dashed red/green line. So this concept might have to be changed.

```
<xsd:complexType name="Shape" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="sl2:SBase">
      <xsd:attribute name="id" type="sl2:SId"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Shape2D" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="sl2:Shape">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Shape3D" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="sl2:Shape">
    </xsd:extension>
  </xsd:complexContent>
```

```
</xsd:complexType>

    .
    .
    .
```

# Render Groups

A render group consists of one or more references to shapes defined in the list of shapes. The ShapeReference objects has a reference to the shape's id and and a transformation to be used on the shape. There can be two types of transformations. Either the user gives the whole matrix for an affine transformation or he specifies the transformation in term of translation, rotation and scaling where tx,ty,tz are the term for translation, rx,ry,rz are the terms for rotation and sx,sy,sz are the terms for scaling. The rotation angles is given in radien.

```
<xsd:complexType name="Transformation">
  <xsd:complexContent>
    <xsd:extension base="sl2:SBase">
      <xsd:attribute name="tx" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="ty" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="tz" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="rx" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="ry" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="rz" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="sx" type="xsd:double" use="optional" default="1.0"/>
      <xsd:attribute name="sy" type="xsd:double" use="optional" default="1.0"/>
      <xsd:attribute name="sz" type="xsd:double" use="optional" default="1.0"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="AffineTransformation">
  <xsd:complexContent>
    <xsd:extension base="sl2:SBase">
      <xsd:attribute name="a0" type="xsd:double" use="optional" default="1.0"/>
      <xsd:attribute name="a1" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="a2" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="a3" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="b1" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="b2" type="xsd:double" use="optional" default="1.0"/>
      <xsd:attribute name="b3" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="b4" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="c1" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="c2" type="xsd:double" use="optional" default="0.0"/>
      <xsd:attribute name="c3" type="xsd:double" use="optional" default="1.0"/>
      <xsd:attribute name="c4" type="xsd:double" use="optional" default="0.0"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ShapeReference">
  <xsd:complexContent>
    <xsd:extension base="sl2:SBase">
    <xsd:sequence>
       <xsd:choice>
         <xsd:element name="transformation" type="sl2:Transformation"/>
```

```
        <xsd:element name="transformation" type="sl2:AffineTransformation"/>
       </xsd:choice>
    </xsd:sequence>
      <xsd:attribute name="shape" type="sl2:SId"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="RenderGroup">
 <xsd:complexContent>
  <xsd:extension base="sl2:SBase">
   <xsd:sequence>
    <xsd:element name="shape" type="sl2:ShapeReference" minOccurs="1" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="id" type="sl2:SId"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

# Connection to Layout

As described in the documentation to the layout part of this extension. We described, that the layout contains compartmentGlyphs, reactionGlyphs and speciesReferenceGlyphs as well as additionalGraphicalObjects. In order to link the render information to the layout, all those objects could be extended be an attribute called renderGroup which would reference the id of the corresponding renderGroup. Additionally we would add a transformation object to all the glyphs mentioned above as to allow a transformation to be used on the renderGroup. We put this transformation here rather than in the renderGroup itself because this would allow the user to use the same render group with several glyphs but at different sizes, rotations and positions etc.

```
<listOfLayouts>
    .
    .
    .
    <speciesGlyph id="ATP_Glyph" species="ATP" x="20.0" y="10.0" w="50" h="50" renderGroup="specRefRender2">
    </speciesGlyph>
    <speciesGlyph id="ATP_Glyph" species="ATP" x="70.0" y="80.0" w="100" h="100" renderGroup="specRefRender1">
        <transformation xsi:type="sl2:Transformation mx="10" my="10" sx="0.8" sy="0.8"/>
    </speciesGlyph>
    .
    .
    .
</listOfLayouts>
<render>
    <listOfColors>
        <color id="Color_Green" red="0" green="255" blue="0"/>
        <color id="Color_Black" red="0" green="0" blue="0"/>
    </listOfColors>
    <listOfLinetypes>
        <lineType id="SolidLine" width="1">
            <strokeType xsi:type="sl2:SolidStroke"/>
        </lineType>
    </listOfLinetypes>
    <listOfShapes>
        <shape xsi:type="sl2:Circle" id="StandCircle" lineType="SolidLine" lineColor="Color_Green" radius="1.0"/>
        <shape xsi:type="sl2:textLabel" id="label1" font="Courier" fontsize="12" text="Glucose" color="Color_Black"/
```

```
        <shape xsi:type="sl2:textLabel" id="label2" font="Courier" fontsize="12" text="ATP" color="Color_Black"/>
    </listOfShapes>
    <listOfRenderGroups>
        <renderGroup id="specRefRender1">
            <shape shape="StandCircle">
                <transformation sx="100" sy="100"/>
            </shape>
            <shape shape="label1">
                <transformation mx="10" my="44"/>
            </shape>
        </renderGroup>
        <renderGroup id="specRefRender2">
            <shape shape="StandCircle">
                <transformation sx="50.0" sy="50.0"/>
            </shape>
            <shape shape="label1">
                <transformation mx="7.0" my="19.0"/>
            </shape>
        </renderGroup>
    </listOfRenderGroups>
</render>
```

O.K. As this is a made up example it probably has tons of mistakes, but we hope that it still shows the general principal that we think could be used to connect layout and render information. So what this example is supposed to show is a layout part with (among other things) two speciesGlyphs. Those have different size and different positions. In the render part, we define some colors and a line type. We don't use any fill types in this example. Next come the basic shapes that we will use to build up the render information. Each shape can in principal be used in more than one renderGroup as demonstrated with the circle. Last but not least we define two renderGroups each with a circle and a text label. Since the textLabels have different sizes, the circle has to be scaled to the right size with s transformation. Now we can reference these two renderGroups in our speciesGlyphs from the layout part. There is now a tag called renderGroup and an optional transformation to be used on the renderGroup. While the first renderGroup is rendered unchanged, the second one is scaled down a little bit and then moved as to be in the center of the bounding box of the speciesGlyph again.

# Outlook

We are aware that this is just a first draft of what might one day become the render information part of the SBML layout extension as proposed by us. We think that this can again be used as a starting point and as a base for discussion with everyone from the sbml community who is interested in this extension. With the help of all those people it is our hope that both extension will be in a usable state before long.