TSS Specification

# TSS namespace

TSS has the xmlns namespace xmlns:tss="http://www.phylotastic.org/tss"

# NeXML Annotations

Annotations to nodes and edges are specified in the NeXML file as meta tags, with the following general format:

<meta id="meta1" property="tss:prop\_name" content="prop\_value" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

Some specific examples

<meta id="meta2" property="tss:taxonomic\_level" content="family" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

<meta id="meta3" property="tss:allele\_freq" content="[70,10,20]" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

<meta id="meta4" property="tss:trophic\_level" content="carnivore" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

<meta id="meta5" property="tss:bootstrap" content="70" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

<meta id="meta6" property="tss:extinct" content="" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

To meet notational standards, the id should be unique for every annotation, but is not used as part of the styling. The xsi:type is required, but is also not used and should always be xsi:type=”nex:LiteralMeta”. The datatype is also required, but for simplicity it is always datatype=”xsd:string” (strictly speaking, not all content will be strings, but we’ll let the viewer decide that as necessary). The property value should begin with “tss” to indicate it is a potential styling annotation. The name of the property, which will be used in the tss sheet to indicate classes comes after the colon. Strictly speaking there should be an additional file generated with the NeXML annotations which defines each property, although this is not actually used as part of the styling. Content can be text, number, or an array (specified by []). Content is also optional if one just wants to add an unspecified tag, such as the property listed as extinct. There is no restriction on what sort of content can be annotated to a node or an edge, but some properties only logically belong to one or another and the specific rendering options vary depending on whether the property is on a node or an edge (see below).

Within the tss, a specific annotation is referred to by its property name (without the css). To refer to only a specific value of a property, use . notation. Thus

trophic\_level

would refer to any node/edge annotated with a trophic\_level, while

trophic\_level.herbivore

would refer to only those with trophic levels whose content is “herbivore”. For numeric values for which one wants to specify a range of content, use the following notation

bootstrap[min=0][max=50]

to specify bootstrap properties with values between 0 and 50 (inclusive).

The tss file itself is set up exactly like a css file to allow existing css parsers to work on it. While much of the tss is the same as in css, some of it is unique to drawing trees. It is up to the tree renderer to know how to use the results, but the tokenization should be handled by any css parser without modification.

Below are described the tss components and properties.

# General Components

## Canvas

The canvas represents the rendering box for the tree. This is functionally similar to the body of an html document.

canvas {

background-color: white;

/\* one might consider supporting other background elements, such as

images \*/

height: 100%;

width: 100%;

/\* the size of the canvas in which the tree will be drawn. For many

viewers the default is the window size, but this would only

specification of both larger and smaller areas as necessary. Allowable

values should be interpreted through general css parameters, including

%, pixels, inches, etc. \*/

font-family: arial, sans-serif;

font-size: 1em;

font-style: roman;

font-weight: bold;

/\* These set the default font properties for the tree. There are no

special parameters, but should support all font options normally

available for CSS \*/

}

## Tree

The tree represents the entire drawn tree and contains default properties for the rendering of the tree as a whole

tree {

border-width: 1px;

border-color: black;

border-style: solid;

/\* the border describes the line styles for the tree. We are using border

since this is a standard notation in css. The shortcut notation of

border: 1px black solid;

should also be supported \*/

layout: rectangular;

/\* layout describes the shape/style of the tree to be drawn. Supported

options are: rectangular | triangular | radial | polar \*/

tip-orientation: right

/\* describes the direction the tree will be drawn, specifically which side

of the canvas should have the tips. Has no meaning for radial and polar

trees. Valid options are: left | right | top | bottom \*/

scaled: true;

/\* describes whether the tree should be scaled to branch lengths are drawn

in an ultrametric/equal branch-length style. Valid options are: true |

false \*/

}

## Scale

Describes the style of a scale bar for the tree

scale {

visible : true;

/\* describes whether a scale bar is to be drawn. Valid options are true |

false \*/

font-family, font-size, etc.

/\* options to override the default fonts set by the canvas /\*

border-color: black;

border-size: 1px;

border-style: solid;

/\* options to control the style of the scale line \*/

scale-width: *value*

/\* specifies the width of the scale bar in scale units (i.e., 1 would

indicate 1 substitution per site. Other than a number, the word “whole”

will indicate that the scale runs the entire length of the tree \*/

scale-title: “*text*”

/\* specifies the title of the scale bar. it must be in quotes and pair of

blank quotes can be used to indicate no title \*/

/\* Additional options needed: control of tick marks? different font options

for title vs. tick labels? \*/

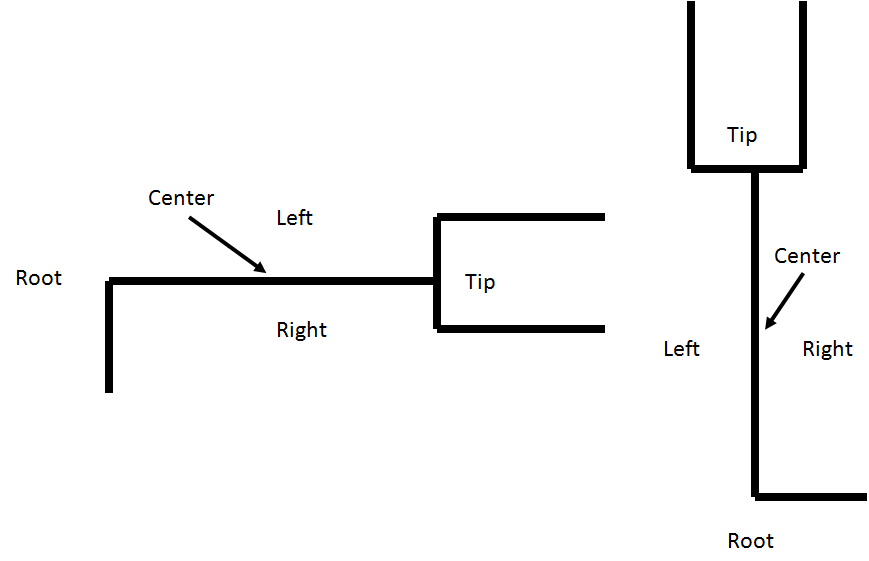
}

# Specific Annotated Components

Everything else in the tss file refers to specific annotations of nodes or edges. The options are detailed below.

Some general properties and parameters will apply to many features. The first of these is the (potential) alignment of the styled element. Where the item should be rendered relative to the annotated element is controlled by the “align” parameter. Because directions on a tree vary relative to the direction it is drawn, we are using the terms “root” and “tip” to specify alignments on the root side or tip side of an object, and “left” and “right” to indicate the left or right side of the object WHEN FACING THE TIP (as opposed to the actual screen). “Center” specifies the center of the element.

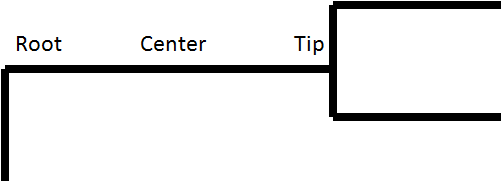
The following figure illustrates the orientation of the align parameter relative to an edge on trees oriented with tips on the right and the top. The same logic works relative to a node.



align: center;

/\* Supported options are: center | tip | root | left | right \*/

When align is left or right for annotated edges, an additional parameter “edge-align” is used to control where along the edge the item is rendered. The options for this are “root”, “tip”, and “center”, as show in the following figure already using “align: left”:



edge-align: center;

/\* Supported options are: center | tip | root \*/

In addition to a few specialized parameters, tree renderers will need to recognize existing css definitions for things such as colors, borders, fonts, and opacity.

## LABELS

Labels for edges and nodes may come from multiple places. First, if the node is an OTU, it may have a label in the OTU block of the NeXML file which specifies a name (e.g., the species name). Specific annotations may contain content which can be used as a label. Finally, the text for a label can be specified based in the tss itself.

To label an edge or node with text based on a specific annotation, use the “text” parameter. Text can either be specified with a literal string in quotes, or be the content of the annotation by using VALUE, or be the label associated with the element by using LABEL. Generally, only OTU’s will make use of the LABEL element. The relative alignment of the text to the annotated element is controlled using the align (and if applicable, edge-align) parameters. Other aspects of the text, such as color, font, etc., use standard css formatting. If not specified, the default font properties are those of the canvas (e.g., by default most tree renderers will label tips with the OTU labels using the default canvas font). Some specific examples of controlling labels with tss:

### Example: labeling nodes with the specified property content

***NeXML annotation:***

<meta id="meta1" property="tss:clade\_b" content="Clade Ruelliae" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

clade\_b {

text: VALUE;

font-family: Helvetica, sans-serif;

font-size: 300%;

color: #000000;

background-color: #FFFFFF;

font-weight: bold;

align: root;

}

### Example: labeling nodes with custom text for a specific property content

***NeXML annotation:***

<meta id="meta1" property="tss:labeled\_clade" content="ruelliae" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

labeled\_clade.ruelliae {

text: "Clade Ruelliae";

color: #00135F;

align: left;

edge-align: center;

}

### Example: labeling edges with bootstrap values above 95% with an \*

***NeXML annotation:***

<meta id="meta1" property="tss:bootstrap" content="98" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

bootstrap[min=95][max=100] {

text: "\*";

font-size: 300%;

align: right;

edge-align: center;

}

### Example: labeling edges with bootstrap values below 50% with specific text

***NeXML annotation:***

<meta id="meta1" property="tss:boostrap" content="47" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

bootstrap[min=1][max=50] {

text: "<50";

font-size: 300%;

align: right;

edge-align: center;

}

### Example: labeling edges with their bootstrap values

***NeXML annotation:***

<meta id="meta1" property="tss:boostrap" content="86" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

bootstrap[min=1][max=50] {

text: VALUE;

font-size: 300%;

align: right;

edge-align: center;

}

### Example: changing the color of OTU labels based on an annotation

***NeXML annotation:***

<meta id="meta1" property="tss:trophic\_level" content="herbivore" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

trophic\_level\_herbivore {

text: LABEL;

color: red;

}

## EDGES

The rendering of the edge itself can be specified for specific annotation by specifying border properties. Examples:

### Example: styling edges with bootstrap values above 95%

***NeXML annotation:***

<meta id="meta1" property="tss:bootstrap" content="98" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

bootstrap[min=95][max=100] {

border-width: 4px;

border-color: red;

}

### Example: styling edges with different content

***NeXML annotation:***

<meta id="meta1" property="tss:trophic\_level" content="herbivore" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

trophic\_level.herbivore {

border-color: green;

}

trophic\_level.carnivore {

border-color: red;

}

### Example: styling edges which have a property

***NeXML annotation:***

<meta id="meta1" property="tss:extinct" content="" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

extinct {

border-color: silver;

border-style: dotted;

}

## Pie Charts

To draw a pie chart, use the special style parameter called pie\_chart, whose only option is VALUE. Generally, the data from the pie chart should be stored as a vector content of the annotation, and should add up to 100 (i.e., the vector content should represent percentages). The colors for each slice are specified using the category\_colors parameter, whose value should be a list of colors in (), one color for each pie slice in the same order as the values in the data vector.

### Example:

The below commands would specify that a pie chart drawn from the allele\_freq annotation should be drawn, centered on the annotated element, with red, green and blue used as the colors for the pie slices, which in the meta annotation example have %’s of 70, 10, and 20 respectively.

***NeXML annotation:***

<meta id="meta1" property="tss:allele\_freq" content="[70,10,20]" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

allele\_freq {

pie\_chart: VALUE;

category\_colors: (red,green,blue);

align: center;

}

## Images and Icons

Images and icons can be added to a node or edge by using the image parameter and specifying the path and name of the image as the value.

### Example: adding images next to nodes based on specific content

***NeXML annotation:***

<meta id="meta1" property="tss:trophic\_level" content="herbivore" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

trophic\_level.herbivore {

image: "path\plant.png";

align: tip;

}

trophic\_level.carnivore {

image: "path\meat.png";

align: tip;

}

### Example: adding images next to edges based on an annotated property

***NeXML annotation:***

<meta id="meta1" property="tss:extinct" content="" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

extinct {

image: "path\skull.gif";

align: right;

edge-align: center;

}

## Symbols

Symbols are best displayed by leveraging the ability to add text labels consisting of special unicode characters. such as • ● ▲ ► ▼ ◄ ♦ ♥ ◊ ■ and ▪. By controlling the size and color of these and similar symbols, one can easy draw circle, boxes, ticks, and other symbols on top of nodes or along edges.

### Example: adding red circles to nodes representing gene duplication events

***NeXML annotation:***

<meta id="meta1" property="tss:split\_type" content="duplication" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

split\_type.duplication {

text: "●";

font-size: 500%;

color: red;

align: center;

}

### Example: adding tick marks and crosses to edges representing character change and reversal

***NeXML annotation:***

<meta id="meta1" property="tss:char1change" content="forward" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

<meta id="meta2" property="tss:char1change" content="reverse" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

char1change.forward {

text: "▌";

font-size: 500%;

color: green;

align: center;

}

char1change.reverse {

text: "×";

font-size: 500%;

color: green;

align: center;

}

## Collapsing Nodes

To specify that a specific node should be collapsed, set the collapsed parameter to true. In addition to collapsing the node, properties of the collapsed set can be specified. Text labels can be added, with their font properties controlled through standard font- and color settings. The color of the collapsed region is controlled by the border-color, since color and background-color are reserved for the label.

### Example: labeling nodes with custom text for a specific property value

***NeXML annotation:***

<meta id="meta1" property="tss:labeled\_clade" content="Ruelliae" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

labeled\_clade.ruelliae {

collapsed: true;

text: VALUE;

align: tip;

border-color: red;

}

## Confidence Intervals

To specify that a confidence interval around a node (e.g., representing a range of depth), use the special bar parameter. The range of the bar should be specified as a two item vector in the content of the property. Since the bar can be viewed as a thick border, we use those properties to style it.

### Example: labeling nodes with custom text for a specific property value

***NeXML annotation:***

<meta id="meta1" property="tss:conf\_int" content="(50,70)" xsi:type="nex:LiteralMeta" datatype="xsd:string"/>

***tss:***

conf\_int {

bar: VALUE;

align: center;

border-color: blue;

border-style: inset;

border-width: 20px;

opacity: 40%;

}