Chart Components

Functional Specifications

Draft 3: October 19, 2005

Abstract

This document describes the functional specifications of the Chart Components improvement for the BIRT 2.0 release.

Document Revisions

Draft	Date	Primary Author(s)	Description of Changes
1	08/25/2005	David Michonneau	Initial Draft
2	10/6/2005	David Michonneau	Added Styles support outside of BIRT
3	10/19/2005	David Michonneau	Modifed Styles API

Contents

1. Styles support

1.1 Introduction

BIRT charts have many elements that are similar to elements in other parts of a report. For example, the text used for a chart title, axis title or legend.

To enable the report developer to create consistent styles for all elements of a report, it should be possible to use BIRT styles for elements on a chart.

Use of styles also allows the developer to leverage other benefits -- such as an update of a style will apply to all text that uses that style

1.2 Use Cases

- Create a report. Add a chart and a label. Notice that the label will show its text in "Serif" font, while the chart uses a Sans Serif font. Create a style called "report". Change its font to Cursive. The label changes its font, but the chart does not. Expected to be able to change the report font globally using a style, including charts. Same is true of font size and color, or any style property.
- Create a report with four charts. Decide to set a common font. Common colors. With anything else in BIRT, you can define a style. But, charts don't support styles, so one must manually set each chart and each property separately. Expected charts to support styles since they are an inherent part of BIRT from the user's perspective.

1.3 BIRT Style support

The chart item will support one report style (defined at the BIRT Report Level), that applies to all of its contents.

1.4 BIRT Predefined Chart Style

One predefined “chart” style will be defined on the chart item extension. This predefined style will be the default style for all chart items, which do not have any style directly set on them

1.5 BIRT Styles hierarchy

The hierarchy is the same as other report items. Here it is as a reminder: the properties are used in the following order, it will go to the next line if the property is undefined at that level.

1- Chart Properties

2- User Style

3- Predefined Chart Style (“chart”)

4- Style of container element of the chart (if any)

5- Predefined Report Style (“report”)

1.6 BIRT Style properties support

Not all style properties are supported by charts. Here is a table showing what is supported, and what it applies to inside the chart:

Style Property	Supported	Applies to

Style Property	Supported	Applies to
Font	Yes (all)	All labels
Background	Color and Image only	Chart background
Text Block	No	N/A
Box	Yes	Chart item
Border	Yes	Chart item
Format number	No	N/A
Format Datetime	No	N/A
Format String	No	N/A
Page Break	Before and After only	Chart item
Map	No	N/A
Rules	No	N/A

1.7 Non-BIRT Styles Support

1.7.1 API

The Chart will support styles outside of a BIRT Report. This will allow any application to define its own styles and apply them to the Chart.

An interface is defined to allow the Chart to query the style properties it needs from its container. Here is the specification for this interface:

```
interface IStyleProcessor
{
   IStyle getStyle(StyledComponent style, Chart chart)
}
interface IStyle
{
   FontDefinition getFont();
   ColorDefinition getBackgroundColor();
   Image getBackgroundImage();
}
```

A new method is added to the Chart model to hook the style processor. Once this processor is set, any attempt to get a style-related property through the chart model will involve this processor in case no value was set. So this processor is providing styles capabilities at the model design layer, and not only during the rendering. Therefore, it's possible to inspect a style-enabled chart model, for instance in a chart builder user interface.

```
interface Chart
{
   public void setStyleProcessor( IStyleProcessor );

}
```

The StyledComponent class allows the style processor to define styles for different parts of the chart. Here is table describing how it works:

StyledComponent constant	Supported Version	Applies to
CHART_ALL : int	2.0	All chart elements
CHART_TITLE : int	2.1	Chart title
CHART_BACKGROUND : int	2.1	Chart background
PLOT_BACKGROUND : int	2.1	Plot Background
LEGEND_BACKGROUND : int	2.1	Legend Background
LEGEND_LABEL : int	2.1	Legend Labels
DATA_LABEL : int	2.1	Data Labels
AXIS_TITLE : int	2.1	Axis Titles
AXIS_LABEL : int	2.1	Axis Labels
AXIS_LINE : int	2.1	Axis Lines
SERIES_TITLE : int	2.1	Series Titles
SERIES_LABEL : int	2.1	Series Labels
VALUES : List	2.1	List of StyledComponents values

1.7.2 CSS Example

In this example, let's assume the chart is used in standalone mode, and included in a CSS environment. CSS styles can be applied by reference (define a style with a name and properties), or inline (define directly the style properties on the element). So here are the necessary integration steps to apply CSS style on a chart:

1- Write a StyleProcessor implementation, that can process CSS styles for a given chart into an IStyle. This processor implementation can then be used for any chart. The processor must be able to handle the cascading logic and determine which css style is applied to the chart.

2- Before rendering or designing the chart, setStyleProcessor() must be called on the Chart model to register the CSS Style Processor, then the chart can be inspected or rendered using the style processor.

2. Labels Overlap

X Labels can quickly overlap when too many points are being plotted. This is a very frequent scenario. There are three solutions to that problem:

2.1 Labels Rotation

This is already available. The user can set a rotation angle on the labels to avoid them overlapping.

2.2 Label Staggering

This is typically a manual option set by the user. The labels will appear on two lines alternatively, reducing the incidence of overlap. This only applies if the labels are shown horizontally. The model API already supports this, but the charting engine is not using it yet.

2.3 Automatic Label drop

The last solution when the first two solutions are not used or not sufficient, is to drop some of the X labels (but no points). The engine will drop some X-Axis labels, trying to drop as few as possible. As there is no meaning to have overlapping labels, this will always be active.

3. Legend Title

3.1 Feature Description

https://bugs.eclipse.org/bugs/show_bug.cgi?id=101667

this feature provides the ability to specify a legend title in the legend block section, with its relative position and font attributes.

3.2 API Change

See new schema here: https://bugs.eclipse.org/bugs/attachment.cgi?id=25600

There is a new Title and TitlePosition attribute in the Legend ComplexType:

```
<xsd:complexType name="Legend">
 <xsd:complexContent>
  <xsd:extension base="Block">
   <xsd:sequence>
    …
    <xsd:element name="Title" type="component:Label" minOccurs="0">
     <xsd:annotation>
      <xsd:documentation>A label instance to hold attributes for legend  title.
       </xsd:documentation>
     </xsd:annotation>
    </xsd:element>
    <xsd:element name="TitlePosition" type="attribute:Position" minOccurs="0">
     <xsd:annotation>
      <xsd:documentation>Specifies where the title for the legend should be  displayed.
       </xsd:documentation>
     </xsd:annotation>
    </xsd:element>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

4. Marker Shapes

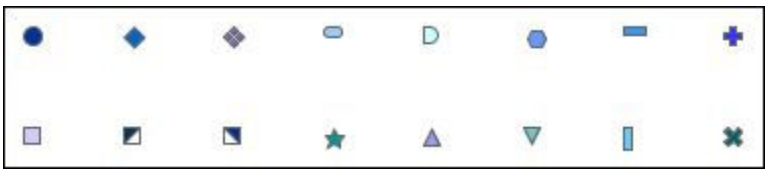
4.1 Bugzilla Entries

https://bugs.eclipse.org/bugs/show_bug.cgi?id=102397

4.2 Feature Description

This will provide a palette of marker icons in addition to the existing marker types (square, circle, triangle, crosshair). The Markers will be able to use that palette of icons to render each serie (there is no “per category” option in that case), so that each point representing a serie will use a given icon from the marker palette.

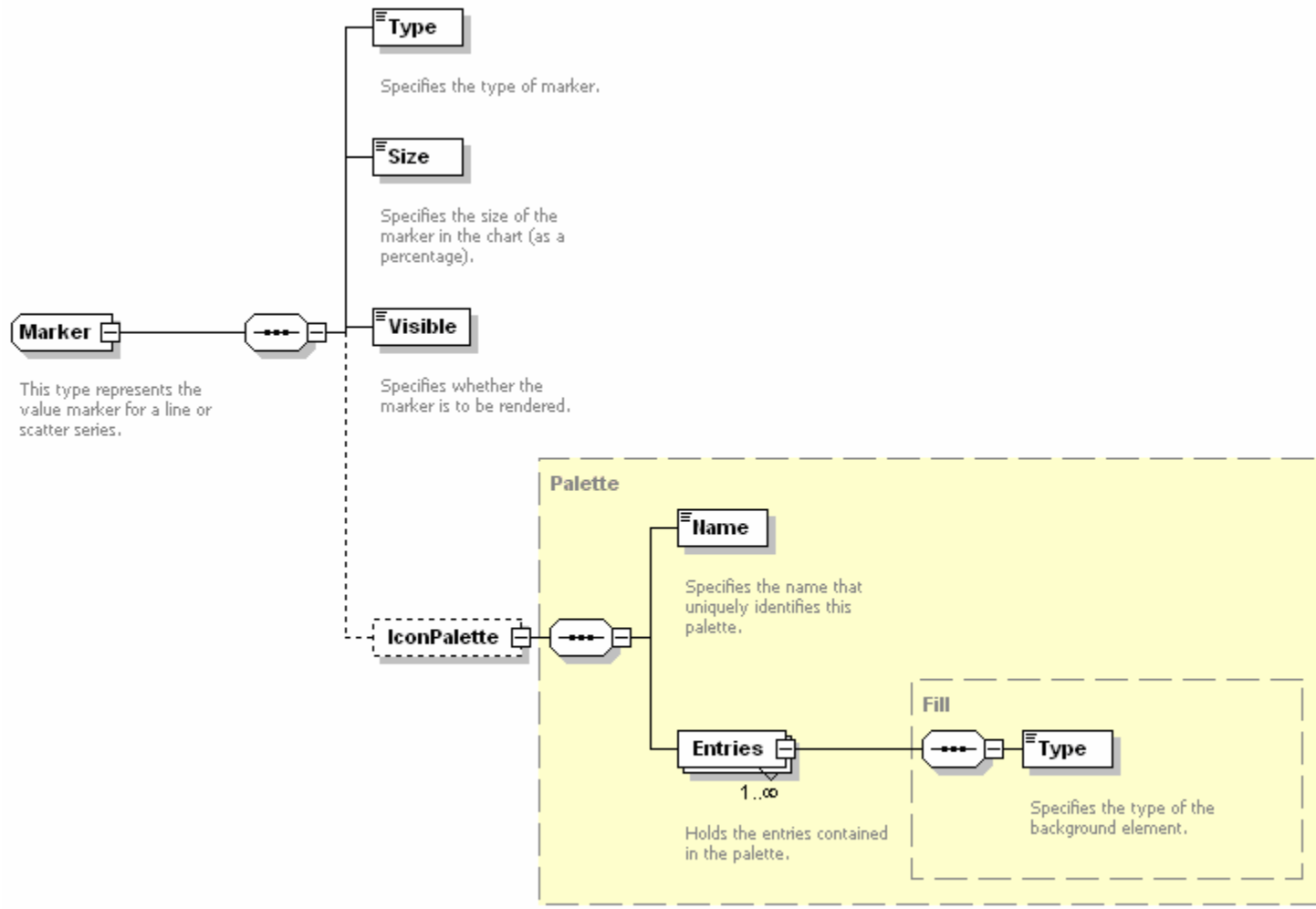
Here are the supported icons that will be provided in the chart builder:



4.3 API Change

Since the Palette already supports images, we only need to associate a separate Palette with the Markers. So that the Markers and the Series can each have their own Palette (in the current model, they share it).

The Marker Type will be extended to support one more type: Icon. In that case, the size will be ignored and the marker will use the icon from the IconPalette, a new Palette attribute on the Marker object. The schema for Marker then becomes as follows:




```
<xsd:simpleType name="MarkerType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        This type represents the possible values for markers supported for Line Series.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Crosshair"/>
        <xsd:enumeration value="Triangle"/>
        <xsd:enumeration value="Box"/>
        <xsd:enumeration value="Circle"/>
        <xsd:enumeration value="Icon"/>
    </xsd:restriction>
</xsd:simpleType>


<xsd:complexType name="Marker">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        This type represents the value marker for a line or scatter series.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="Type" type="MarkerType">
            <xsd:annotation>
```

```
                    <xsd:documentation xml:lang="en">
                    Specifies the type of marker.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Size" type="xsd:int">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Specifies the size of the marker in the chart (as a percentage).
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Visible" type="xsd:boolean">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Specifies whether the marker is to be rendered.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="IconPalette" type="Palette" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```