Maongo Documentation

Release 0.7.1

4=1

February 14, 2012

CONTENTS

Note: Stand: February 14, 2012

Note: *PDF-Version <pdf/Maongo.pdf>*

Section author: jo *Section author: jo*

CHAPTER

ONE

FÜR WEN IST DAS BOOK OF MAONGO?

Das Book of Maongo wendet sich an den Autor multimedialer, interaktiver Anwendungen. Uns ist bewusst, dass Maongo neu ist, dass wir sowohl den Funktionsumfang als auch die Sprachelemente und die Konzepte, die Maongo einzigartig machen, erläutern müssen.

Was also wollen wir dem Autor nahebringen?

- XML-Sprachumfang

  Wir stellen die XML-Elemente vor, aus denen die Maongo Application Description (MAD) besteht. Wir orientieren uns dabei an den Widget-Typen, aus denen das Framework besteht (Kapitel 4 bis 7).

- JavaScript

  In allen Kapiteln wird auf die Nutzung der vorgestellten Features in "Actions", also JavaScript-Methoden, eingegangen. Maongo-spezifische JavaScript-Konstrukte sind an zwei Händen abzuzählen (Kapitel 3); der Rest ist pures JavaScript.

- Grundkonzepte

  Wir stellen Grundkonzepte vor, die für die Arbeit mit Maongo essentiell sind (Kapitel 2, 3 und 8 bis 13). Zum Beispiel Kapitel 8 "Data und Routen": Maongo bietet einzigartige Features, um Daten zu laden und sie durch die Anwendung zu senden, wo sie als Trigger für Aktionen, als Datenobjekte, an die sich GUI-Elemente binden können, oder zum Umschalten von `Cards` in einem `StackWidget` dienen können.

Darüber hinaus bietet Kapitel 14 eine Übersicht aller Maongo-Bauelemente und ihrer Properties; die Kapitel 15 und 16 behandeln die Maongo-Infrastruktur wie die MaongoPlayer-Applikation oder die Integration von Maongo-Anwendungen in Adobe Flex.

CHAPTER

TWO

DER MAONGOPLAYER / MAONGO-INSTALLATION

Der MaongoPlayer dient zur Ansicht und zum Auspielen von Maongo-Presentations.

- Download der OSX-Application: http://www.maongo.com/osx-download.html
- Download als JAR (plattformübergreifend, Java 1.6): http://www.maongo.com/jar-download.html

Für die Flash-Ausspielung benötigen Sie zusätzlich das Adobe Flex SDK, das sie gratis von folgender URL herunterladen können:

- http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3 (Adobe Flex SDK, v. 3.4)

Bitte geben Sie im MaongoPlayer (Menü Export > Select Flex SDK) den Speicherort des SDKs auf Ihrem Rechner an!

CHAPTER

THREE

DAS MAD-DOKUMENT

Maongo-Präsentationen (Presentations) werden in der Beschreibungssprache Maongo Application Description (MAD) verfasst. MAD ist eine deklarative XML-Sprache, die das Layout, die Hierarchie, die Logik, die Animationen u.a.m. einer Presentation beschreibt.

Ein MAD-Dokument trägt die Endung `.mad`.

Innerhalb eins MAD-Dokuments kann Javascript als Scripting Sprache verwendet werden. Siehe dazu :ref:‘ref-actions

Die meisten Sachverhalte in Maongo lassen sich sowohl in XML und als auch in JavaScript ausdrücken. Die Dokumentation wird daher beide Schreibweisen präsentieren.

Namen und Bezeichner sind in Maongo case-sensitiv.

3.1 Presentation

Die Presentation bildet die Grundfläche, auf der weitere Widgets (s.u.) platziert werden. Die Presentation ist ein Widget mit speziellen Eigenschaften und kann nur als äußerer Container benutzt werden. Eine Presentation ist immer rechteckig. Die Größe der Fläche wird über die `width`- und `height`-Attribute definiert.

Eine “leere” Presentation kann wie folgt definiert werden:

```
<presentation width="400" height="300">
        <property name="Title" value="Mein erstes Maongo"/>
</presetation>
```

Die Presentation besitzt die Property `Title`, mit der der zu verwendende Fenstertitel gesetzt werden kann. Weiterhin besitzt eine Presentation auch alle anderen Eigenschaften eines Widgets, wie beispielsweise `BackgroundColor` oder `Rotation`.

Die Presentation ist der einzige Ort, in der die der Anwendung zur Verfügung stehenden Fonts deklariert werden können (s.u. *Schriften*). Außerdem ist die Presentation der bevorzugte Ort, vererbbare Properties auf einen Standardwert zu setzen: beispielsweise die Farbangaben (`BackgroundColor`, `ForegroundColor`).

Die Presentation kann in allen Widgets mit `this.Presentation` (JavaScript) bzw. durch Lookup auf “Presentation” adressiert werden.

3.2 Widgets

Ein Widget ist der Grundbaustein im Maongo System. Es ist eine graphische Komponente, die eine Reihe von veränderbaren Eigenschaften (Properties) besitzt. Das Widget kann Daten empfangen, verarbeiten und weiterleiten, es ist

durch Funktionen (Actions) erweiterbar, es lässt sich animieren und in automatischen Layouts verwenden.

Widgets werden auf der Fläche der `<presentation>` angeordnet und können ineinander verschachtelt werden. Es entsteht so ein hierarchischer Widget-Baum, der die Maongo-Anwendung ausmacht. Befinden sich mehrere Widgets auf derselben hierarchischen Ebene, so werden sie in der Reihenfolge der Definition gezeichnet. Das zuerst (zuoberst im Quellcode) definierte Widget liegt demnach in der hintersten Zeichenebene.

Das Maongo-Framework enthält unterschiedliche Widgets für verschiedene Einsatzzwecke (Text, Button, Charts, ...), die alle Erweiterungen der Klasse `Widget` sind.

Die Widgets sind organisiert in den Sammlungen *Maongo-Toolkit*, *Maongo-Charts* und *Maongo-Media*.

3.3 Engines

Bestimmte Funktionen benötigen keine grafische Komponente; sie sind in Maongo als Engines realisiert. Eine Engine wird wie ein Widget im XML definiert, hat aber keine visuelle Repräsentanz. Beispiele für Engines sind die Kommunikations-Engines im *Maongo-Toolkit* und der ChartController in *Maongo-Charts*.

3.4 Includes

Der Maongo-Code muss nicht in einem einzigen Dokument stehen. Der Include-Mechanismus erlaubt es, Anwendungen modular zu organisieren und mehrfach benötigten Code an verschiedenen Stellen einzubinden.

Das oberste XML-Element einer Include-Datei ist `<mad>`. Eine Include-Datei kann weitere Dateien inkludieren.

Eine Include-Datei wird mit mit dem Element `<include source="*URL*"/>` inkludiert, das Attribute “source” wird als absolute oder zum inkludierenden Dokument relative URL angegeben.

Die folgende Presentation inkludiert zwei MAD-Dateien:

```
<presentation width="800" height="450">
  <widget>
    <include source="test_1.mad"/>
  </widget>
  <widget>
    <include source="test_2.mad"/>
  </widget
</presentation>


// Quellcode der Datei test_1.mad
<mad>
  <widget>
     <property name="BackgroundColor" value="red"/>
  </widget>
</mad>

// Quellcode der Datei test_2.mad
<mad>
  <widget>
     <property name="BackgroundColor" value="green"/>
  </widget>
</mad>
```

Der Include-Mechanismus wird direkt beim Laden der Presentation ausgeführt; der Parser löst die `<mad />`-Tags auf und fügt den Inhalt der inkludierten Datei an Stelle des `include`-Statements ein.

3.5 Templates

Eine andere Möglichkeit, wiederverwendbare Objekte zu definieren, bilden Templates. Ein Template ist ein Widget (u.U. mit Child-Widgets), das zur späteren Verwendung definiert, aber nicht dargestellt wird.

Zur Definition wird der `<template>`-Tag verwendet. Alle Attribute und Properties sind anolog zum Widget zu verwenden. Templates müssen ein gültiges `name`-Attribut haben:

```
<template name="meinTemplate"/>
```

Zur Verwendung des Templates an anderer Stelle im XML definieren Sie ein Widget und setzen als `type`-Attribut den Templatenamen mit vorangestelltem `@`:

```
<widget type="@meinTemplate" />
```

Properties, die bereits im Template definiert sind, dürfen an dieser Stelle nicht noch einmal definiert werden. Soll der Property-Wert des Templates überschrieben werden, so ist der <set>-Tag zu nutzen.

Ein einfaches Beispiel:

```
 <!-- Template wird definiert -->
<template name="BoxTemplate">
  <property name="Shape" value="R 30,30"/>
</template>

<!-- Template wird mehrfach verwendet und um unterschiedliche
Hintergrundfarben ergänzt -->
<widget name="Box1" type="@BoxTemplate" x="10" y="10">
  <property name="BackgroundColor" value="red"/>
</widget>

<widget name="Box2" type="@BoxTemplate" x="50" y="10">
  <property name="BackgroundColor" value="green"/>
</widget>
```

In einem komplexeren Beispiel werden im Template ein Vielzahl von Properties vordefiniert:

```
<!-- Template wird definiert -->
<template name="HeadLineTemplate" type="Text">
  <property name="MultiLine" value="true"/>
  <property name="TextAlign" value="top-left"/>
  <property name="ForegroundColor" lookup="Design.HeadlineColor"/>
  <property name="BackgroundColor" value="transparent"/>
  <property name="Texture"         lookup="HeadlineTexture"/>
  <property name="TextureMode"     value="force"/>
</template>

<!-- Template wird verwendet -->
<widget name="HeadLine" type="@HeadLineTemplate" x="0" y="0" shape="R 614 100">
  <property name="Font" value="Swiss721BT-Bold-32"/>
  <property name="BaseLineOffset" value="-8"/>
  <property name="LineHeight" value="40"/>
  <set property="BackgroundColor" value="red" />
</widget>
```

Im Beispiel wird eine Headline mit Farben und Hintergründen im Template definiert. Positionierung und Größe der Headline werden erst bei Verwendung des Templates gesetzt.

Templates können in der gezeigten Art und Weise im XML verwendet werden; mit JavaScript lassen sich auch Widgets zur Laufzeit aus Templates erzeugen.

Beispiel:

```
var newWidget = Type.addTemplateToWidget(myParent, "BoxTemplate", "NewBox");
```

3.6 Benennung von Widgets

Widgets (und andere Elemente wie Templates und Actions) können aus anderen Widgets oder aus Skrtipten heraus adressiert werden, wenn sie ein `name`-Attribut haben. Der Name eines Maongo-Objekts darf aus Buchstaben, Zahlen und Unterstrich bestehen:

```
Gültige Zeichen innerhalb eines Widget Namens: [a-zA-Z0-9_]+
```

Der Widget-Name ist stets case-sensitiv.

Der Widget-Name wird im Namespace seines Parent-Widgets verwaltet. Bei Doppelungen innerhalb des gleichen Namespace (z.B. wenn eine Property, ein Define und/oder ein Widget dieselbe Bezeichnung haben), wirft der MAD-Parser einen Fehler.

Fehlerhaftes Beispiel:

```
<presentation>
 <widget name="Box1">
   <property name="BackgroundColor" value="red"/>
 </widget>

 // dies führt zu einem Namenskonflikt
 <widget name="Box1">
   <property name="BackgroundColor" value="green"/>
 </widget>
</presentation>
```

Funktionierendes Beispiel:

```
<presentation>
  <widget name="Box1">
    <property name="BackgroundColor" value="red"/>
  </widget>

  <widget name="Box2">
    <property name="BackgroundColor" value="green"/>
    // aufgrund der Verschachtelung kann der Name "Box1" hier wieder genutzt werden
    <widget name="Box1">
      <!--...-->
    </widget>
  </widget>
 <presentation>
```

Benannte Objekte werden beim Lookup ohne weitere Pfadangaben gefunden, wenn der Name im Namespace des Widgets selbst, des Parents, seines Parents, (etc. bis zur Presentation) gefunden werden kann (vgl. *Defines, Lookups, Bindings*).

3.7 Definition von externen Ressourcen

Wenn in einem Export extern abgelegte und in der MAD-Datei referenzierte Daten wie bpsw. Bilder etc. berücksichtigt und ggf. an einen anderen Ausspielort mitkopiert werden sollen, so müssen die entsprechenden Verzeichnisse/Dateien in der Presentation als `<resource>` gekennzeichnet werden.

Beispiel:

```
<presentation>
   <resource directory="./assets"/>
   <resource file="./assets/animage.png"/>

   <!-- ... -->
</presentation>
```

3.8 MAD-Validierung

Die formale Gültigkeit von MAD-Dateien kann mit einer XML Schema Definition (XSD) überprüft werden. Manche Texteditoren können die XSD-Datei auch für Code Completion nutzen.

Um Validierung und ggf. Code Completion nutzen zu können, empfehlen wir folgenden Dokument-Kopf:

```
<?xml version="1.0" encoding="UTF-8"?>
<presentation xmlns="http://www.maongo.com/mad"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.maongo.com/mad/mad.xsd"
              >
  <!-- ... -->
</presentation>
```

Ist kein Online-Zugriff auf die XSD-Datei gewünscht, so kann sie auch neben das mad-Dokument gelegt und die `schema-location` wie folgt modifiziert werden:

```
xsi:schemaLocation="http://www.maongo.com/mad/mad.xsd mad.xsd"
```

CHAPTER

FOUR

DATENTYPEN

Section author: jo

Note: XXXXXXXXXXXXXXX Status 12.10.2010: verbindlich. XXXXXXXXXXXXXXXXXXXX

fehlt: JS-Beispiel für myList.iterator()

TODO: Javascript und Zugriffe für Compound Datatypes: Ist-Zustand dokumentieren

- Ergänzungen JavaScript-Syntax
- Test, ob die XML- und JavaScript-Konstrukte funktionieren (z.B. define name="xxx" type="Table[String]" value="[[a,b,c],[d,e,f]]"></define>)
- Unittests ergänzen
- Fehlende Types ergänzen (sh. mp.types): Time, ...

4.1 Datentypen in XML und JavaScript

Alle Datentypen werden im XML als Strings repräsentiert. Sie in den richtigen Datentyp zu wandeln, übernehmen in MaongoMP sogenannte Transcoder.

Im XML ist dieser Vorgang automatisch, häufig ist noch nicht einmal eine `type`-Angabe nötig. So können Property-Werte ohne Type gesetzt werden, da das Widget "weiß", von welchem Typ seine Properties sind. Auch Defines können ohne `type`-Angabe auskommen und speichern so zunächst einen Wert vom Typ String. Wird er einer Property zugewiesen, so wird er automatisch transcodiert.

In JavaScript kann ein Objekt eines bestimmten Typs mit folgender Syntax erzeugt werden:

```
Type.newInteger(42)             // ohne Transcoding
Type.newRectangle(100,100)
Type.newShape("R 0,0,100,100")  // Ausnahme; Transcoding nötig //Geht nicht, CF
etc.
```

Ebenfalls möglich ist die folgende Schreibweise, die immer zwei String-Argumente erwartet und wie die Formulierung im XML immer mit einem Transcoding-Vorgang einhergeht:

```
$$("Integer", "42")
$$("Rectangle", "0,0,100,100")
$$("Shape", "R 0,0,100,100")
etc.
```

4.2 Auflistung der Datentypen

Object

Ein generisches Objekt. Alle weiteren Datentypen erben von Object; für den Anwender nicht direkt zu nutzen. Nicht zu verwechseln mit dem `<object>`-Tag, der immer eine `type`-Angabe erfordert und somit für ein spezifisches (typisiertes) Objekt steht.

Integer

Eine generische Ganzzahl, positiv oder negativ, mindestens 32 Bit breit.

XML:

```
<... type="Integer" value="42"/>
```

Number

Eine generische Fließkommazahl, positiv oder negativ, mindestens 32 Bit breit.

XML:

```
<... type="Number" value="42.23"/>
```

Decimal

In Java als BigDecimal implementiert. Andere Plattformen nutzen u.U. Fließkommazahlen.

XML:

```
<... type="Decimal" value="42.5"/>
```

Boolean

Folgende Schreibweisen sind gültig: `true`, `false`.

String

Ein generischer String.

Symbol

Spezieller Datentyp z.B. für Properties, bei denen nur bestimmte Begriffe als Werte erlaubt sind.

Konvertierbarkeit: Alle Typen die nach String konvertierbar sind, können auch nach Symbol konvertiert werden.

XML:

```
<... type="Symbol" value="top-left"/>
```

Date

Datentyp für Datums- und Zeitangaben

XML:

```
<... type="Date" value="2010-01-01" />
<... type="Date" value="2010-01" />
<... type="Date" value="2010" />
<... type="Date" value="2010-01-01 00:15:24" />
```

Point

Eine Koordinate auf einer Fläche. Die X- und Y-Werte der Koordinate werden als zwei durch Komma oder Whitespace getrennte Numbers angegeben.

XML:

```
<... type="Point" value="10.2,15.4"/>
```

Padding

Padding beschreibt den Abstand zweier ineinanderligender Objekte an den vier Seiten oben, unten, rechts und links. Padding kann mit einer Wertangabe (alle Abstände gleich), mit zwei (oben/unten und links/recht) oder vier (oben, rechts, unten, links) Werten definiert werden:

XML:

```
<... type="Padding" value="23.4"/>
<... type="Padding" value="23.4,42.3"/>
<... type="Padding" value="10,15,12,5"/>
```

Die Widget-Property Padding wird im Abschnitt *Position, Größe und Form* beschrieben.

Color

Der Datentyp Color beschreibt eine Farbe mit den Komponenten Rot, Grün, Blau und Alpha. Die Alphawerte beschreiben die Opazität der Farbe (entsprechend der Verwendung in CSS); der Wert 0 weist die Farbe als vollständig transparent aus, der Wert 1 (oder 255 / FF) als vollständig opak. Ist kein Alpha-Anteil angegeben, so gilt die Farbe stets als vollständig opak.

Farbangaben mit Farbnamen

Maongo unterstützt die 16 Farbnamen der VGA-Palette sowie die Bezeichner “orange” und “transparent”. Die Farbnamen werden klein geschrieben, die Schreibweise ist case-sensitiv.:

```
* black   = #000000
* silver  = #C0C0C0
* gray    = #808080
* white   = #FFFFFF
* maroon  = #800000
* red     = #FF0000
* purple  = #800080
* fuchsia = #FF00FF
* green   = #008000
* lime    = #00FF00
* olive   = #808000
* yellow  = #FFFF00
* navy    = #000080
* blue    = #0000FF
* teal    = #008080
* aqua    = #00FFFF
* orange  = #ffa500
* transparent = #000000/00
```

Hexadezimale Farbangaben in HTML-Schreibweise

Farbwerte können hexadezimal nach dem Langschema #RRGGBB oder dem Kurzschema #RGB angegeben werden. Bei der Kurzschreibweise werden die Angaben intern verdoppelt. “#FCF” steht dann für “#FFCCFF”. Die hexadezimale Schreibweise ist case-insensitiv.

```
#RRGGBB
#RGB
```

Hexadezimale Farbangaben mit Alpha

Folgt nach der Hexadezimalzahl ein Slash "/", werden die beiden nächsten hexadezimalen Ziffern als Angaben für den Alphawert gewertet. Die Kurzschreibweise #RGB/A ist ebenfalls möglich. Die hexadezimale Schreibweise ist case-insensitiv:

```
#RRGGBB/AA oder #RGB/A
```

Beispiele:

```
#008000                 Grün, 100% opak ( == #008000/FF )
#008000/00              Grün, 100% transparent
#008000/FF              Grün, 100% opak
#000000/00 = "transparent"
```

Numerische Farbangaben nach CSS mit RGB(A)

Eine weitere Möglichkeit, Farben anzugeben, ist die rgba-Notation. Die Werte werden numerisch von 0 bis 255 angegeben.

Der Wert "a" definiert die Opazität, ist fakultativ und wird von 0 bis 1 angegeben, wobei 1 für 100 % Deckkraft steht:

```
rgb(0,255,0)            Grün, 100% opak
rgba(0,255,0,1)         Grün, 100% opak
rgba(0,0,0,0) = "transparent"
```

File

Der Datentyp `File` erwartet einen String (ohne Protokoll) für eine Pfadangabe im lokalen Filesystem. Die Pfadangabe muss absolut formuliert sein. Es findet keine Übersetzung der Plattform-Spezifika statt.

URL

Der Datentyp `URL` erwartet einen URI (mit Protokoll http oder file, absolut oder relativ) als String.

Image

Der Datentyp `Image` erwartet einen Lademodus und einen URL (mit Protokoll http oder file, absolut oder relativ) als String in eckigen Klammern.

Gleichbedeutend sind:

```
<property name="Texture" type="Image" value="http://my.server.com/images/picture.png" />
<property name="Texture" type="Image" value="[lazy http://my.server.com/images/picture.png]" />
```

Als Angaben für den Lademodus sind erlaubt:

- `lazy` - der Ladevorgang wird angestossen, sobald das Bild verwendet wird
- `preload` - Das Bild wird geladen, bevor die Presentation gestartet wird
- `embed` - Das Bild wird - wenn auf der jeweiligen Plattform möglich - mit in die Presentation integriert. Es wird geladen, bevor die Presentation gestartet wird

Alternativ kann auch nur der URL angegeben werden, der Lademodus ist dann implizit `lazy`.

Media

Der Datentyp `Media` erwartet eine Map zur Definition der verschiedenen Eigenschaften.

Einfache Syntax:

```
<property name="Source" type="Media" value="URL:http://meinserver.de/v1.mp4"/>
```

Um detaillierte Informationen zu einer Source liefern zu können, kann auch eine explizit definierte Map übergeben werden

Beispiel einer RTMP-Source:

```
<property name="Source">
  <map>
    <entry name="RTMPServerPath">rtmp://mediaserver.de/_definst_</entry>
    <entry name="RTMPMediaPath" >rtmp_stream_flv</entry>
    <entry name="ConnectionType" >rtmp</entry>
    <entry name="UseFCSubscribe">true</entry>
  </map>
</property>
```

Zur Beschreibung des Mediums können die folgenden Einträge in der Map übergeben werden:

URL (String) Die Url von der das Medium wiedergegeben werden soll. Es können HTTP, RTMP oder RTMPT-Quellen angegeben werden. Bei Angabe einer HTTP-Quelle ist die Angabe der Property `MediaPath` nicht notwendig. Für RTMP- bzw. RTMPT-Quellen ist hier die URL des Media-Servers anzugeben. Die Angabe des wiederzugebenden Mediums erfolgt in der Property `MediaPath`

Hinweis:

```
RTMP-Quellen werden derzeit nur in der Flash-Ausspielung unterstützt. RTMPT-Quellen werden n
```

MediaPath (String) Sofern RTMP/RTMPT-Quellen genutzt werden sollen, muss in dieser Property der für den Verbindungsaufbau notwendige Pfad zur tatsächlichen Medienquelle angegeben werden.

ConnectionType (String) Art der Verbindung die hergestellt werden soll.

Default: `http`

Values: `http`, `rtmp`, `rtmpt`

UseFCSubscribe (String) Soll für die Verbindung mit einem Media-Server über RTMP/RTMPT ein Subscribe Aufruf geschickt werden? Dies ist bei bestimmten Media-Servern notwendig um Medien abspielen zu können.

Default: `false`

Values: `true`, `false`

siehe: *MediaPlayerWidget*

Shape

`Shape` ist ein Interface, das durch eine Reihe von Klassen (`Rectangle`, `Shape`, `Path`, `Polygon`, `Circle`, `Ellipse`) implementiert wird.

Properties, die Werte vom Datentyp `Shape` erwarten, können mit allen vom Interface abgeleiteten Datentypen beschickt werden.

Rectangle

Der Typ `Rectangle` erlaubt die Definition von Rechtecken in der folgenden Form:

```
"(px py )w h"
"0 0 100 100"
"100 100"
"10 10 100 100"
"-10 -10 100 100"
Verwendung: <property name="Shape" type="Rectangle" value="0 0 100 100" />
oder <widget rect="0 0 100 100" />
```

Die Angabe `px py` bezeichnet den Punkt, an dem die linke obere Ecke des Rechtecks im Koordinatensystem des Widgets gezeichnet wird; sie ist optional. Je nach Kontext kann die Angabe von negativen Werten in `px py` invalide sein (zum Beispiel bei der Maongo-Property `InnerShape`).

w und h geben die Breite und Höhe des Shapes in Pixeln an.

Eine alternative Schreibweise für Rectangle ist, den Typ `Shape` anzugeben und den Wert als String so zu formulieren:

```
"R (px py )w h"
"R 0 0 100 100"
Verwendung: <property name="Shape" type="Shape" value="R 0 0 100 100" />
```

Path

Freigeformte Shapes werden in SVG-Pfadsyntax angegeben und sind vom Typ `Path`. Der folgende SVG-Pfad:

```
<path d="M 20 20 L 18 22 C 24 28 14 25 10 40 Q 20 45 15.33 60" />
```

wird als Shape-Beschreibung so genutzt:

```
<property name="Shape" type="Path" value="M 20,20 L 18,22 C 24,28 14,25 10,40 Q 20,45 15.33,60"
```

MaongoMP unterstützt alle in der SVG 1.1-Spezifikation für Pfade definierten Elemente (http://www.w3.org/TR/SVG11/paths.html). Wo eine Ausspielplattform für bestimmte Kurventypen keine native Unterstützung bietet, werden diese näherungsweise gezeichnet.

Polygon, Kreis, Ellipse

Die SVG-Syntax für Polygone, Kreise und Ellipsen können bei Angabe des entsprechenden Typs verwendet werden:

```
SVG: <polygon points="100 100 100 200 150 200" />
Verwendung: <property name="Shape" type="Polygon" value="<x y> <x y>..." />
<property name="Shape" type="Polygon" value="100,100 100,200 150,200" />
alternative Schreibweise:
<property name="Shape" type="Shape" value="P 100,100 100,200 150,200" />

SVG: <circle cx="100" cy="100" r="50" />
Verwendung: <property name="Shape" type="Circle" value="<cx> <cy> <r>" />
<property name="Shape" type="Circle" value="100 100 50" />
alternative Schreibweise:
<property name="Shape" type="Shape" value="O 100 100 50" />

SVG: <ellipse cx="100" cy="100" rx="50" ry="20" />
Verwendung: <property name="Shape" type="Ellipse" value="<cx> <cy> <rx> <ry>" />
<property name="Shape" type="Ellipse" value="100 100 50 20" />
```

```
alternative Schreibweise:
<property name="Shape" type="Shape" value="E 100 100 50 20" />
```

Maongo akzeptiert alle SVG-tauglichen Schreibweisen für die Shape-Werte. Werte können beispielsweise durch Leerzeichen, Komma oder auch gemischt getrennt werden:

```
<property name="Shape" type="Polygon" value="100 100 100 200 150 200" />
<property name="Shape" type="Polygon" value="100,100 100,200 150,200" />
<property name="Shape" type="Polygon" value="100,100,100,200,150,200" />
```

4.3 Compound Datatypes

List Im XML des MAD-Dokuments können auch (ein- und mehrdimensionale) Listen definiert werden. Das Trennzeichen zwischen Listenitems ist stets das Semikolon `;`. Im `type`-Attribut kann die Liste typisiert werden. Ist kein Typ angegeben, so wird angenommen, dass die Listenitems Strings sind:

```
<... type="List[Number]" value="3,4,3.5" />
<... type="List[String]" value="Hans,Dieter,Peter" />
<... type="List" value="Hans,Dieter,Peter" />
```

Ein besonderer Fall ist der Typ `List[KeyValue]`, der beispielsweise zur Definition eines LayoutRasters benutzt wird:

```
<... type="List[KeyValue]" value="fl: 10, s1: 4, chart: max, ll: 0" />
```

Jedes Listenitem besteht hier aus einem `key` (`fl`) und einem Wert (`10`), getrennt durch Doppelpunkt.

Listen können auch in einer expliziten Syntax beschrieben werden:

```
<list>
  <item type="Number">3</item>
  <item type="Number">4</item>
  <item type="Number">3.5</item>
</list>
<list>
  <item>Hans</item>
  <item>Dieter</item>
  <item>Peter</item>
</list>
```

Eine zweidimensionale Liste in verschiedenen Syntax-Varianten:

```
<property name="Values">
  <list>
    <list>
  <item type="Number">3</item>
    <item type="Number">4</item>
    <item type="Number">3.5</item>
  </list>
  <list>
  <item type="Number">5</item>
  <item type="Number">6</item>
  <item type="Number">6</item>
  </list>
```

```
  </list>
</property>

oder:

<property name="Values">
  <list>
    <item type="List[Number]">3,4,3.5</item>
    <item type="List[Number]">5,6,6</item>
  </list>
</property>

oder:

<property name="Values" type="List[List[Number]]"  value="[[3,4,3.5],[5,6,6]]" /> //Geht nicht,
```

Beispiele für Listenzugriffe mit Lookups:

```
<define name="one" type="Number" value="1"/>
<define name="two" type="Number" value="2"/>
<define name="examplelist" type="List[Number]" value="[1,2,3]"/>
<define name="examplelist" type="List[Number]" value="1,2,3"/>

<property type="List" value="[@one,@two]"/>
( ==> [1,2] )
<property type="List[Number]" value="@examplelist"/>
( ==> [1,2,3] )
<property type="List" value="[@examplelist]" />
( ==> [[1,2,3]] )
<property type="List[Number]" value="[@examplelist,2,@examplelist]"/>
( ==> [[1,2,3],2,[1,2,3]] )
<property type="List" value="[@examplelist,2,@examplelist]"/>
( ==> [[1,2,3],"2",[1,2,3]] - die Elemente der äußeren Liste werden wo nötig als String typisier
<property type="List[Number]" value="@examplelist,2,@examplelist"/>
( ==> null - eckige Klammern nötig ) //CF: Bzw. wirft eine NullPointerException und macht damit
```

Zugriff auf Lists:

JavaScript:

Neue (leere) Liste erzeugen:

```
var myList = Type.newList();

// auch: var myList = $$("List");
```

Methoden von List nutzen:

```
myList.add(17);
trace(myList.size());
// 1
myList.add(21);
trace(myList.get(0));
// 17
myList.remove(0);
trace(myList.get(0));
// 21
```

JavaScript-Arrays können in Listen umgewandelt werden *NYI*:

```
var jsArray = [1, "hans", 13];
var myList = Type.newListFromArray(jsArray); //CF geht im moment nicht. Bug. Also schon implemen

trace(myList.size());
// 3
trace(myList.get(0));
// 1
trace(myList.get(1));
// "hans"
```

Map Maps sind ungeordnete Sammlungen von Key/Value-Paaren. Der Zugriff erfolgt nur über den Key; die Einträge haben keine spezifische Reihenfolge. Entries sind typisiert.

XML-Struktur:: <define name="myMap"> <map> <entry name="aKey" value="aValue"/> <entry name="aKey2" value="aValue2"/>

</map>

</define>

Zugriff auf Maps:

```
fehlt
```

JavaScript:: Eine Map wird als maongo.core.ValueMap zurückgeliefert. Auf diese Map kann man mit map.get("key") oder map.getValueForKey("key") zugreiffen.

var myMap = $(this,"mapDefinition"); trace(myMap.getValueForKey("aKey")); // >> aValue

Table Ein Table ist eine geordnete Sammlung von `Rows`, die wiederum eine geordnete Sammlung von (typisierten) Cells enthal
//CF: Table kann nicht definiert werden

XMLStruktur:

```
<table>
  <row>
    <cell>
```

Zugriff auf Tables:

```
fehlt
```

JavaScript:

```
fehlt
```

Textformat

XML-Struktur:

```
<textformat>
  <property>
```

JavaScript:

```
fehlt
```

Textformate können nicht in JavaScript modifiziert werden.

Vgl. den Abschnitt *TextWidget*

Data

Ein Data-Objekt ist eine Sammlung von `Property`, `Table`, `Map` und `List`-Objekten.

XML-Struktur:

```
<data>
  <property />
  <table />
  <map />
  <list />
</data>
```

Zugriff auf Data:

```
fehlt
```

JavaScript:

```
fehlt
```

Transmission

Eine Transmission ist eine Sammlung von `Data`-Objekten.

XML-Struktur:

```
<transmission>
  <data />
  <data />
</transmission>
```

JavaScript:

```
fehlt
```

CHAPTER

FIVE

DAS WIDGET

Ein Widget ist der Grundbaustein im Maongo System. Es ist eine graphische Komponente, die eine Reihe von veränderbaren Eigenschaften (Properties) besitzt, die Daten empfangen und weiterleiten kann, die durch Funktionen (Actions) erweiterbar ist, und die sich animieren lässt. Es existieren unterschiedliche Widgets mit unterschiedlichen Eigenschaften für spezifische Einsatzzwecke (siehe die folgenden Kapitel).

Das einfachste Widget ist quadratisch (100x100) und hat seinen Ursprung (0,0 Koordinate des Widgets) in der oberen linken Ecke seines Parents:

```
<widget name="ErstesWidget" type="Widget" />
```

Das zweite Widget wird 10 Einheiten (Pixel) vom linken und 20 Einheiten vom oberen Rand des übergeordneten Widgets platziert. Es hat eine rechteckige Form und ist 400x300 Einheiten gross. Die Angabe des Typs "Widget" ist optional und kann weggelassen werden:

```
<widget name="ZweitesWidget" type="Widget" x="10" y="20" shape="R 400 300" />
```

Die Attribute `x` und `y` bezeichnen die Position des Widgets im Koordinatensystem des übergeordneten Widgets ("parent"). Das Attribut `shape` erlaubt, die Gestalt des Widgets zu setzen (hier in Form eines Rechtecks).

Die interne Struktur von Widgets lässt sich in Form mehrerer Ebenen beschreiben:

```
(hinten)
Füllebene       (Hintergrundfarbe)
Texturebene     (Bild als Texture)
Zeichenebene    (widgetspezifische Inhalte, z.B. Text, Video, ...)
Kinderebene     (alle Child-Widgets)
Rahmenebene     (Border)
(vorn)
```

In einer Anwendung werden Widgets häufig hierarchisch ineinander verschachtelt. Vom Root-Knoten aus lässt sich dann ein Widget-Baum mit parent-child-Relationen beschreiben:

```
<presentation>
  <widget name="widget1">
    <widget name="widget2" />
    <widget name="widget3" />
  </widget>
</presentation>
```

widget1 ist Kind (`child`) von presentation; widget2 und widget3 sind Kinder von widget1. Entsprechend ist widget 1 `parent` von widget2 und widget3.

Befinden sich mehrere Widgets auf derselben hierarchischen Ebene, bestimmt die Definitionsreihenfolge auch die Zeichenreihenfolge: Die zuerst definierten Widgets liegen "hinter" den später definierten Widgets: widget2 liegt hier hinter widget3.

5.1 Allgemeine Properties

Jedes Widget verfügt über eine Vielzahl von Properties, welche die Darstellung bzw. das Verhalten des Widgets in der Anwendung beeinflussen. Diese Properties können mittels des XML-Befehls:

```
<property name="PROPERTYNAME" value="PROPERTYVALUE"/>
```

oder:

```
<property name="PROPERTYNAME">PROPERTYVALUE</property>
```

gesetzt werden (vgl. auch *Das Type-Value-Prinzip*)

Parallel dazu können die Properties auch aus Scriptbereichen in der Anwendung (vgl. *Actions*) gesetzt werden. Im folgdenden wird auf diese Möglichkeit nur an ausgewählten Stellen eingegangen. Die generelle Syntax für den Zugriff auf Properties in Scripten ist:

```
<action>
   // Zugriff auf eine Property des Widget auf dem diese Action aufgerufen wird
   this.BackgroundColor = "red";

   // Zugriff auf eine Property über eine Referenz auf ein anderes Widget welche
   // in der Variablen "myWidget" gespeichert ist
   var myWidget = lookup("OtherWidget");
   myWidget.BackgroundColor = "green";

</action>
```

5.1.1 Benennung von Widgets

Name (String) Die Benennung von Widgets mit der Property `Name` ist in den meisten Fällen nicht zwingend nötig, aber immer empfohlen, da es die Übersicht und das Debugging erleichtert. In folgenden Fällen **müssen** Sie die Widgets benennen:

- wenn Sie per Lookup (im XML oder JavaScript) auf ein Widget oder seine Children zugreifen wollen
- wenn Sie per JavaScript ein Widget (oder seine Children) steuern wollen
- wenn Sie ein Template definieren (Templates werden über den Namen referenziert)
- wenn Sie Funktionalität nutzen wollen, die auf Widgets mit vordefiniertem Namen angewiesen sind (z.B. das `widget name="Grid"` im LineChart)

Default: `""`

Wählen Sie den Namen so, dass er nicht mit einer Widget-Property oder selbst definierten Variablen kollidiert. Im Namensraum eines Widgets darf eine Bezeichnung nicht mehrfach vorkommen.

Title (String) Diese Property kann nur auf der Presentation gesetzt werden. Sie wird, wo es die Ausspielplattform erlaubt, angezeigt (z.B. als Fenstertitel im Projektor und im Browser).

Default: `"No Title"`

5.1.2 Spezielle Properties

Presentation (Widget) In dieser Property ist in jedem Widget eine Referenz auf die Presentation, also die oberste Hierarchiestufe der Anwendung gepspeichert.

Base (URL) In der `Base` wird die Basis-URL der Anwendung gespeichert, welche genutzt wird um bspw. Pfadangaben für Bilder aufzulösen. Diese Property ist nur auf der Presentation vorhanden.

Default: `undefined`

IO (Object) Diese Property beinhaltet eine Referenz auf das Input/Output (IO) System der Anwendung. Dieses kann genutzt werden um bspw. Tastaturen oder andere Eingabegeräte anzusprechen. Diese Property ist nur auf der Presentation vorhanden.

Children (List[Widget], read-only) Diese Property beinhaltet eine Liste der `Children` des Widgets.

5.1.3 Debugging

Debug (Boolean) Schaltet Debug-Informationen ein, es werden der Ankerpunkt, die Bounds und Padding markiert.

Default: `false`

Werte: `true`, `false`

5.1.4 Data

vgl. den Abschnitt *Data und Routen*

Data (Data)

DataObserver (Data)

5.1.5 Layout

Vgl. den Abschnitt *Layout*

Layout (Symbol)

LayoutConstraints (String)

PreferredBounds (Rectangle)

5.1.6 Animation

Vgl. den Abschnitt *Animation*

AnimationControl (Symbol) Dient zum Starten/Stoppen der auf dem Widget definierten (Haupt-)Animation.

Default: `""`

Werte: `play`, `pause`, `stop`, `rewind`, `playfromstart`

```
<beispiel hier/>
```

5.2 Position, Größe und Form

5.2.1 Location (Point)

Default: `0,0`

Die Eigenschaft `Location` verankert das Widget im Koordinatensystem des Elternwidgets:

```
<property name="Location" value="16 9.2" />
```

Jedes Widget besitzt ein Koordinatensystem, das von links nach rechts (x-Achse) und von oben nach unten (y-Achse) wächst. Eine Einheit im Koordinatensystem entspricht im Normalfall (Bildschirmausgabe, Widget ist nicht skaliert) einem Pixel. Ein Widget kann auch auf Teilpixeln positioniert werden, also beispielsweise bei 9.2 Pixeln.

Die Eigenschaft `Location` wird mit dem Typ `Point` angegeben. Der Standardwert der Eigenschaft `Location` ist `"0 0"`, die Nullpunkte der Koordinatensysteme der Eltern- und Kinder-Widgets liegen also per Default übereinander.

Angabe der Location im Widget-Tag

Die Eigenschaft `Location` kann auch anhand von `x`- und `y`-Attributen im Widget-Tag angegeben werden:

```
<widget x="16" y="9.2">
   <property .../>
</widget>
```

Die beiden Schreibweisen schließen sich gegenseitig aus. Sollten Sie sowohl Angaben zur Location im `widget`-Tag als auch mit einem `property`-Tag machen, wird der Maongo-Player einen Fehler ausgeben.

5.2.2 X (Number, read-only)

In dieser Eigenschaft ist die aktuelle horizontale Position des Widgets im Koordinatensystem des Parent-Widgets gespeichert.

5.2.3 Y (Number, read-only)

In dieser Eigenschaft ist die aktuelle vertikale Position des Widgets im Koordinatensystem des Parent-Widgets gespeichert.

5.2.4 Z-Index (Number)

Default: `-1` (nicht aktiv)

Mittels dieser Eigenschaft ist ein manuelle Sortierung der Widgets in der Z-Achse möglich. Sofern ein Z-Index gesetzt wird, gilt nicht mehr die Reihenfolge der Widgets im Mad-Dokument, sondern der Z-Index.

5.2.5 Width (Number, read-only)

In dieser Eigenschaft ist die aktuelle Breite des Widgets bzw. der `Bounding-Box` gespeichert.

5.2.6 Height (Number, read-only)

In dieser Eigenschaft ist die aktuelle Höhe des Widgets bzw. der `Bounding-Box` gespeichert.

5.2.7 Shape (Shape)

Default: `R 0 0 100 100`

Werte: `Shape` (SVG(+)-Notation), `Path` (Pfad), `Rectangle` (Rechteck), `Circle` (Kreis), `Ellipse` (Ellipse), `Polygon` (Vieleck) vgl. *Datentypen*

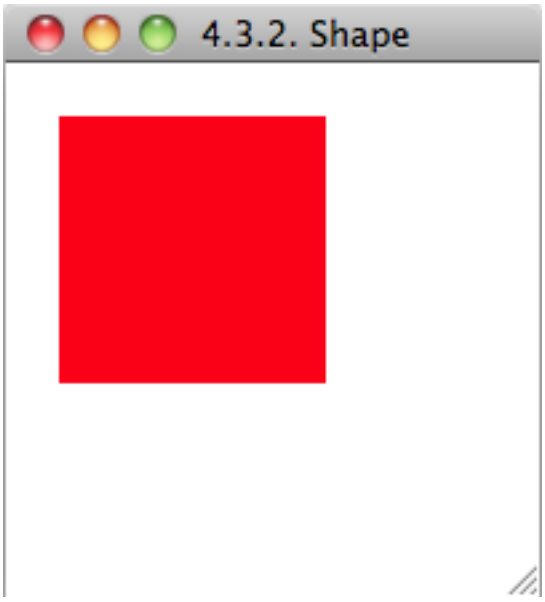
Die Form eines Widgets wird von der Eigenschaft `Shape` bestimmt:

```
<widget shape="R 200 300"></widget>
oder:
<property name="Shape" value="R 200 300" />
oder:
<property name="Shape" value="R 20 20 100 100" />
```

Das Default-Rechteck wird automatisch gezeichnet, sobald ein Widget angelegt wurde und keine Shape-Eigenschaft spezifiziert wird:

```
<widget>
  <property name="Location" type="Point" value="20,20"/>
  <property name="BackgroundColor" value="red"/>
</widget>
```

Das Widget wird mit "Location" an Punkt "20,20" im Koordinatensystem seines Parent positioniert. Es wurde kein Shape spezifiziert, deshalb wird per Default ein Rechteck mit 100 Pixeln Breite und 100 Pixeln Höhe gezeichnet. Im Bild wird das Default-Rechteck durch die rote Hintergrundfarbe (Eigenschaft "BackgroundColor) sichtbar gemacht.



Achtung: Wird das Shape eines Widgets per Script auf ''null' gesetzt, so wird es intern auf "R 0 0 0 0" gesetzt.'

Rechteckige Shapes

Für rechteckige Shapes gibt es mehrere Beschreibungsmöglichkeiten:

```
<widget shape=”R 0 0 100 100” />
<widget>
  <property name="Shape" value="R 0 0 100 100" />
</widget>
```

Die Angabe von "0 0" ist optional.

Allgemeine Formulierung:

```
<property name="Shape" value="R (<x>),(<y>),<width>,<height>" />
```

Beispiel:

```
<presentation>
 <widget name="myWidget">
   <property name="Location" value="20 20" />
   <property name="Shape" type="Shape" value="R 10,30,80,80" />
   <property name="Debug" type="Boolean" value="true" />
   <property name="BackgroundColor" type="Color" value="red" />
 </widget>
</presentation>
```

Der Nullpunkt von myWidget wird am Punkt 20,20 des Elternsystems verankert. Gezeichnet wird das Widget mit einer rechteckigen Form, die im Punkt 10,30 des eigenen Koordinatensystems beginnt und jeweils 80 Pixel in horizontaler und vertikaler Richtung einnimmt.

Frei geformte Shapes

Shapes müssen in Maongo nicht rechteckig sein. Vielmehr können Sie freie Formen annehmen, die in Anlehnung an die SVG-Syntax für Pfad und Polygon, Kreis und Ellipse definiert werden.

Allgemeine Informationen zu „Skalierbaren Vektorgrafiken“ (SVG) finden Sie unter den folgenden URLs:

- http://de.wikipedia.org/wiki/Scalable_Vector_Graphics
- http://www.w3.org/Graphics/SVG/
- http://www.w3.org/TR/SVG11/

Im Folgenden stellen wir SVG-Syntax und Maongo-Schreibweise für die verschiedenen Shape-Typen dar:

Pfad:

```
<!-- SVG: -->
<path d="M 16,23 5,18 -5,24 -4,12 -13,4 -1,1 3,-9 9,1 22,2 13,11 16,23 z" />

<!-- Maongo: -->
<property name="Shape" value="<d>" />

<property name="Shape" value="M 16,23 5,18 -5,24 -4,12 -13,4 -1,1
3,-9 9,1 22,2 13,11 16,23 z" />
```

Polygon:

```
<!-- SVG: -->
<polygon points="100 100 100 200 150 200" />

<!-- Maongo: -->
<property name="Shape" value="P <x y> <x y>..." />

<property name="Shape" value="P 100,100 100,200 150,200" />
```

Kreis:

```
<!-- SVG: -->
<circle cx="100" cy="100" r="50" />

<!-- Maongo: -->
<property name="Shape" value="O <cx> <cy> <r>" /> <!-- der Buchstabe O steht für den Kreis -->

<property name="Shape" value="O 50" />
<!-- der Nullpunkt des umschließenden Rechtecks ist auf 0,0 -->
<property name="Shape" value="O 100,100 50" />
<!-- der Nullpunkt des umschließenden Rechtecks ist auf 100,100 verschoben -->
```

Ellipse:

```
<!-- SVG: -->
<ellipse cx="100" cy="100" rx="50" ry="20" />

<!-- Maongo: -->
<property name="Shape" value="E <cx> <cy> <rx> <ry>" />

<property name="Shape" value="E 100 50" />
<!-- der Nullpunkt des umschließenden Rechtecks ist auf 0,0 -->
<property name="Shape" value="E 100,100 100 50" />
<!-- der Nullpunkt des umschließenden Rechtecks ist auf 100,100 verschoben -->
```

SVG-Werkzeuge

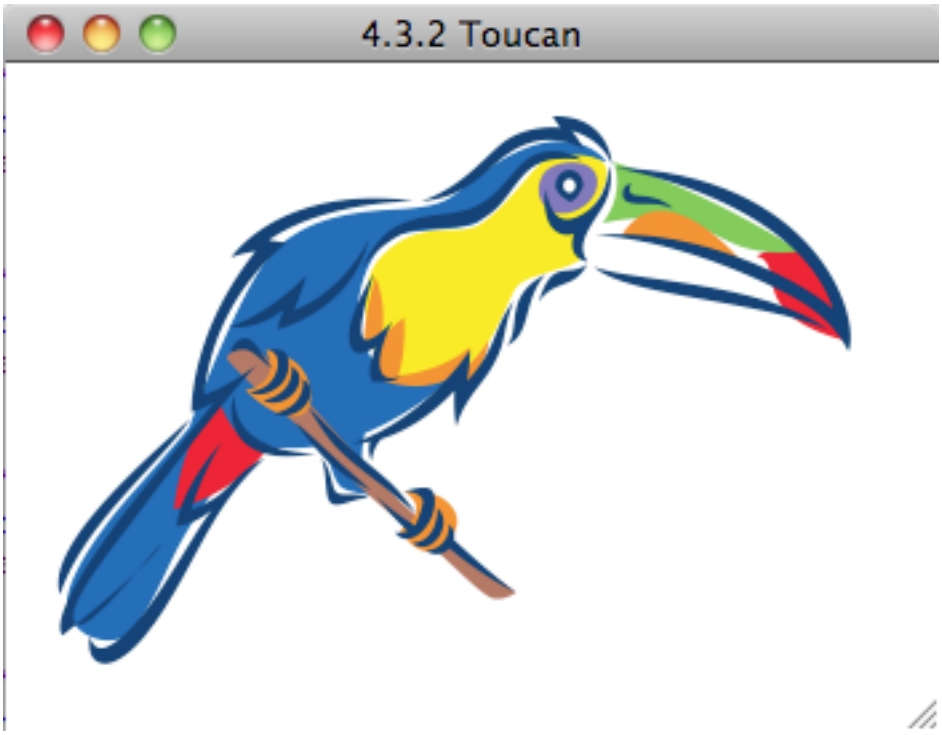
- Inkscape (http://www.inkscape.org) ist ein kostenloser OpenSource-Vektorgrafik-Editor für alle Plattformen.

  Tip: "Als normales SVG" speichern. Die Pfadangaben aus Inkscape können Sie dann direkt in Maongo verwenden.

- Adobe Illustrator (http://www.adobe.com/products/illustrator) ist der Marktführer unter den Vektorgrafik-Programmen. SVG ist ein mögliches Speicherformat.

  Tip: Polygon- und Pfadangaben können Sie direkt aus Illustrator-SVGs in Maongo übernehmen.

Transforms und Style-Angaben aus SVGs können nicht direkt in Maongo übernommen werden. Die entsprechenden Mittel haben Sie in Maongo in Properties wie Location und Rotation u.a.m. SVG-Pfade können aber durchaus für komplexe grafische Elemente verwendet werden.



Das entsprechende **Mad Dokument**.

5.2.8 Padding (Padding)

Default: `null`, Padding ist ausgeschaltet.

Die Property `Padding` dient dazu einen Abstand (Padding) von der Bounding-Box des Shapes nach innen zu definieren. Dieser Abstand wird z.B. von einigen Layouts genutzt, die nur in der vom Padding vorgegebenen Fläche layouten:

```
<property name="Padding" value="20" />
<property name="Padding" value="20 30" />
<property name="Padding" value="10 8.5 20 15" />
```

Die Maongo-Syntax für Padding orientiert sich an der CSS-Syntax. Erlaubt sind 1, 2 und 4 Angaben, die folgendermaßen interpretiert werden:

- Eine Angabe: Alle Innenabstände sind gleich groß.
- Zwei Angaben: Die erste Zahl steht für die Abstände oben und unten, die zweite Angabe für die Abstände rechts und links.
- Vier Angaben: die Abstände werden im Uhrzeigersinn zugeordnet: oben, rechts, unten, links.

Beispiel:

```
<widget name="widget1" shape="R 100 100">

  <property name="Padding" value="10 8 20 5" />
  <property name="BackgroundColor" value="red" />

  <property name="Layout" value="fill" />
  <widget name="widget2">
```

```
    <property name="LayoutConstraints" type="String" value="padding"/>
    <property name="BackgroundColor" type="Color" value="yellow" />
  </widget>

</widget>
```

widget2 wird durch ein Fill-Layout auf die Größe von widget1 (exakt: auf die Größe minus Padding) gebracht. Die Angabe für Padding bewirkt, dass das InnerShape von widget1 modifiziert wird.

5.2.9 InnerShape (Rectangle, read-only)

Die Eigenschaft `InnerShape` ist die Form, die aus der Kombination der Bounding-Box des Shapes und Padding entsteht. Das InnerShape ist stets ein Rectangle. Das `InnerShape` ist bei vielen Widget-Typen und bei Layouts die Begrenzung, innerhalb derer Inhalte dargestellt werden. So wird beispielsweise der Text im Textwidget innerhalb des Innershapes dargestellt.

5.2.10 BorderWidth (Number)

Default: `0` (beim ButtonWidget: `1`)

`BorderWidth` legt die Breite des Rahmens eines Widgets fest:

```
<property name="BorderWidth" type="Number" value="10.6"/>
```

Die Eigenschaft `BorderWidth` ist vom Typ Number. Der Rahmen hat die Standardbreite 0, d.h. ohne die Angabe einer `BorderWidth` gibt es keinen Rahmen. Die Farbe eines Rahmens wird mit der Property `BorderColor` festgelegt.

Beispiel:

```
<widget>
  <property name="Location" type="Point" value="20 20" />
  <property name="Shape" value="R 20 20 100 100" />
  <property name="BorderWidth" value="10.6"/>
  <property name="BorderColor" value="#ffcc00"/>
  <property name="BackgroundColor" value="red"/>
</widget>
```

Der Rahmen wird mittig auf den Rand der Form (Shape) gezeichnet.

5.2.11 Visible (Boolean)

Default: `true`

Werte: `true`, `false`

Mit der Eigenschaft `Visible` legen Sie fest, ob das Widget angezeigt wird oder nicht:

```
<property name="Visible" value="false" />
```

5.2.12 Clipping (Boolean)

Default: `true`

Werte: `true`, `false`

Die Eigenschaft `Clipping` legt fest, ob ein Widget seine Kinder-Widgets am eigenen Rand abschneidet, oder ob die Kinder-Widgets über das Shape des Eltern-Widgets hinausragen dürfen:

```
<property name="Clipping" value="true" />
```

5.2.13 PreferredBounds (Rectangle)

Setzen der Property `PreferredBounds` bewirkt, dass ein Widget innerhalb eines Layouts diesen Wert an seinen Parent zurückliefert, nicht eine selbst bestimmte ideale Größe. Siehe auch *Layout*.

5.2.14 JavaScript

Im folgenden einige beispielhafte Zugriffe und Zuweisungen auf Properties aus diesem Kapitel dargestellt.

Javascript-Syntax:

```
myWidget.Location = "20,20";
myWidget.Shape    = "R 20,20";
myWidget.Shape    = Type.newRectangle(20,20);
myWidget.Shape    = Type.newRectangleAt(5,5,20,20);
myWidget.Shape    = "E 100 50";
myWidget.Shape    = Type.newEllipse(20,30);
myWidget.Padding  = "10 8.5 20 15";
myWidget.Visible  = false;

trace("Widget an: " + myWidget.X + "x" + myWidget.Y)  // ergibt bspw. "Widget an: 100x120"
```

Note: Siehe auch:

CLASSDOC:Shape

CLASSDOC:Point

CLASSDOC:Padding

CLASSDOC:Path

CLASSDOC:Polygon

CLASSDOC:Rectangle

CLASSDOC:Transform

5.3 Farben

Jedes Widget besitzt die Eigenschaften `BackgroundColor`, `ForegroundColor` und `BorderColor`, mit denen Sie dem Widget Hintergrund-, Vordergrund- und Rahmenfarbe zuweisen können.

Diese Widget-Eigenschaften werden vererbt, einen Standard-Wert gibt es nur für das Root-Element `presentation`. Alle Kinder-Widgets erben die Farben aus ihrem Elternwidget, wenn sie sie nicht explizit neu setzen. Farbangaben

können dabei in unterschiedlicher Form erfolgen : als Farbname (“red”), in HTML-Schreibweise (“#ff0000”) oder als rgba-Angabe mit Alpha. Siehe *Datentypen*.

5.3.1 BackgroundColor (Color)

Default: `white`

Die Eigenschaft “BackgroundColor” legt die Hintergrundfarbe eines Widgets fest:

```
<property name="BackgroundColor" value="green" />
```

Die Eigenschaft BackgroundColor ist vom Typ “Color”, der Typ muss in der Definition nicht genannt werden. Das Root-Element “Presentation” hat per Default die Hintergrundfarbe Weiß. Alle weiteren Widgets erben ihre Hintergrundfarbe vom Elternwidget.

Note: Wenn die Ausspielplattform es unterstützt, kann die in der Hintergrundfarbe angegebene Transparenz einer Presentation bis in die Ausspielung durchschlagen (z.B. kann der Painter ein teil-transparentes PNG erzeugen).

5.3.2 ForegroundColor (Color)

Default: `black`

Die Eigenschaft “ForegroundColor” legt die Vordergrundfarbe eines Widgets fest. Im Basis-Widget hat diese Eigenschaft keine Auswirkung. Sie ist für spezialisierte Widgets wie beispielsweise das Text-Widget interessant, bei dem sie die Textfarbe beeinflusst.

```
<property name="ForegroundColor" value="blue" />
```

Die Eigenschaft ForegroundColor ist vom Typ “Color”, der Typ muss in der Definition nicht genannt werden. Das Root-Element “Presentation” hat per Default die Vordergrundfarbe Schwarz. Alle weiteren Widgets erben ihre Vordergrundfarbe vom Elternwidget. Im Basiswidget hat die ForegroundColor keine direkte Auswirkung. In anderen Widgets bestimmt sie das Aussehen des Haupt-Inhaltes; im Textwidget beispielsweise die Farbe des Textes.

5.3.3 BorderColor (Color)

Default: `black`

Die Eigenschaft “BorderColor” legt die Rahmenfarbe eines Widgets fest:

```
<property name="BorderColor" value="red" />
```

Die Eigenschaft BorderColor ist vom Typ “Color”. Das Root-Element “Presentation” hat per Default die Rahmenfarbe Schwarz. Alle weiteren Widgets erben ihre Rahmenfarbe vom Elternwidget. Der Rahmen wird erst sichtbar, wenn die Property BorderWidth auf einen Wert > 0 gesetzt wird.

5.4 Alpha

5.4.1 Alpha (Number)

Default: `1`

Die Property Alpha setzt die Opazität eines Widgets (inklusive seiner Children).

Die Property ist bisher nicht implementiert und wirkt sich nicht aus.

```
<property name="Alpha" value="0.3" />
```

Default: 1 (opak)

Werte: 0 - 1

5.5 Rotation und Skalierung

5.5.1 Rotation

Default: `0`

Die Eigenschaft "Rotation" dreht das Widget um seinen Center (standardmäßig den Punkt `0,0` im widget-eigenen Koordinatensystem):

```
<property name="Rotation" type="Number" value="10"/>
```

Der Drehwinkel wird als Zahl in Grad angegeben. Positive Werte drehen das Widget im Uhrzeigersinn, negative Werte drehen es gegen den Uhrzeigersinn. Der Standardwert der Eigenschaft Rotation ist 0. Der Wert 360 entspricht einer vollständigen Drehung.

Die Eigenschaft Rotation wird auf Kinder-Widgets nicht vererbt. Der Inhalt eines Widgets wird aber mit dem Eltern-Widget gedreht.

Beispiel:

```
<widget name="widget1" x="80" y="20" shape="R 100 100">
  <property name="BackgroundColor" value="#ff0000" />
  <property name="Rotation" type="Number" value="20"/>
  <widget name="widget2" shape="R 30 30">
    <property name="BackgroundColor" value="#ffcc00" />
  </widget>
</widget>
```

Das Widget rotiert um seinen Ankerpunkt im Koordinatensystem. Das Child-Widget widget2 wird mitrotiert.

5.5.2 Scale

Default: `1`

Die Eigenschaft "Scale" verändert die Größe des Widgets:

```
<property name="Scale" value="2"/>
```

Die Eigenschaft Scale wird mit dem Typ Number angegeben, der Standardwert ist 1, also keine Skalierung. Der Inhalt eines Widgets - Video, Text, aber auch seine Child-Widgets - wird mit dem skalierten Widget mitskaliert, die Eigenschaft wird nicht vererbt.

Der Bezugspunkt der Skalierung ist der Center-Punkt (standardmäßig `0,0`) des Widgets.

Setzen der Property `Scale` führt dazu, dass das Koordinatensystem des betreffenden Widgets und seiner Kinder skaliert wird.

Beispiel:

```
<widget name="widget1">
  <property name="Location" type="Point" value="20,20"/>
  <property name="BackgroundColor" value="red"/>
  <property name="Scale" value="2"/>
  <property name="BorderWidth" value="1"/>
  <widget name="widget2" x="10" y="10">
    <property name="BackgroundColor" value="yellow"/>
    <property name="Shape" value="R 50 50"/>
  </widget>
</widget>
```

Note: Auch wenn es grundsätzlich möglich ist, ist es nicht empfehlenswert, die Property `Scale` auf der Presentation selbst zu setzen. Soll die ganze Presentation skaliert werden, empfiehlt es sich, ein Widget um den Inhalt der Presentation zu legen, das nur zur Skalierung dient:

```
<presentation width="800" height="600">
  <widget name="Scaler" shape="R 400 300">
    <property name="Scale" value="2" />
    <property name="BackgroundColor" value="#990000" />

    <!-- Inhalt der Presentation -->
    <widget x="350" y="250" shape="R 50 50">
      <property name="BackgroundColor" value="#009900" />
    </widget>
    <!-- Ende -->

  </widget>
</presentation>
```

5.5.3 Center

Die Eigenschaft "Center" wirkt sich nur im Zusammenhang mit den Eigenschaften Rotation und Scale aus. Mit Center können Sie den Bezugspunkt verändern, um den das Widget gedreht oder skaliert wird:

```
<property name="Center" type="Point" value="20 20"/>
```

Die Eigenschaft Center wird mit dem Typ "Point" angegeben, der Default-Wert ist `0,0`.

Beispiel:

```
<widget name="widget1" x="80" y="80">
  <property name="Shape" value="R 100 100"/>
  <property name="Center" value="50,50"/>
  <property name="BackgroundColor" value="red"/>
  <property name="Rotation" value="10"/>
  <widget name="widget2">
    <property name="Location" value="25,25"/>
    <property name="BackgroundColor" value="yellow"/>
    <property name="Shape" value="R 50 50"/>
  </widget>
</widget>
```

Durch die Angabe eines Center-Punktes ersparen wir uns die Verschiebung des Shapes im Koordinatenraum, auch die Platzierung des Childwidgets ist intuitiver.

5.5.4 JavaScript

Im folgenden einige beispielhafte Zugriffe und Zuweisungen auf Properties aus diesem Kapitel dargestellt.

Javascript-Syntax:

```
myWidget.Center   = Type.newPoint(50,50);
myWidget.Scale    = 2.0;
myWidget.Rotation = 10;
```

5.6 Texturen

5.6.1 Texture

Die Eigenschaft "Texture" weist einem Widget eine Textur zu:

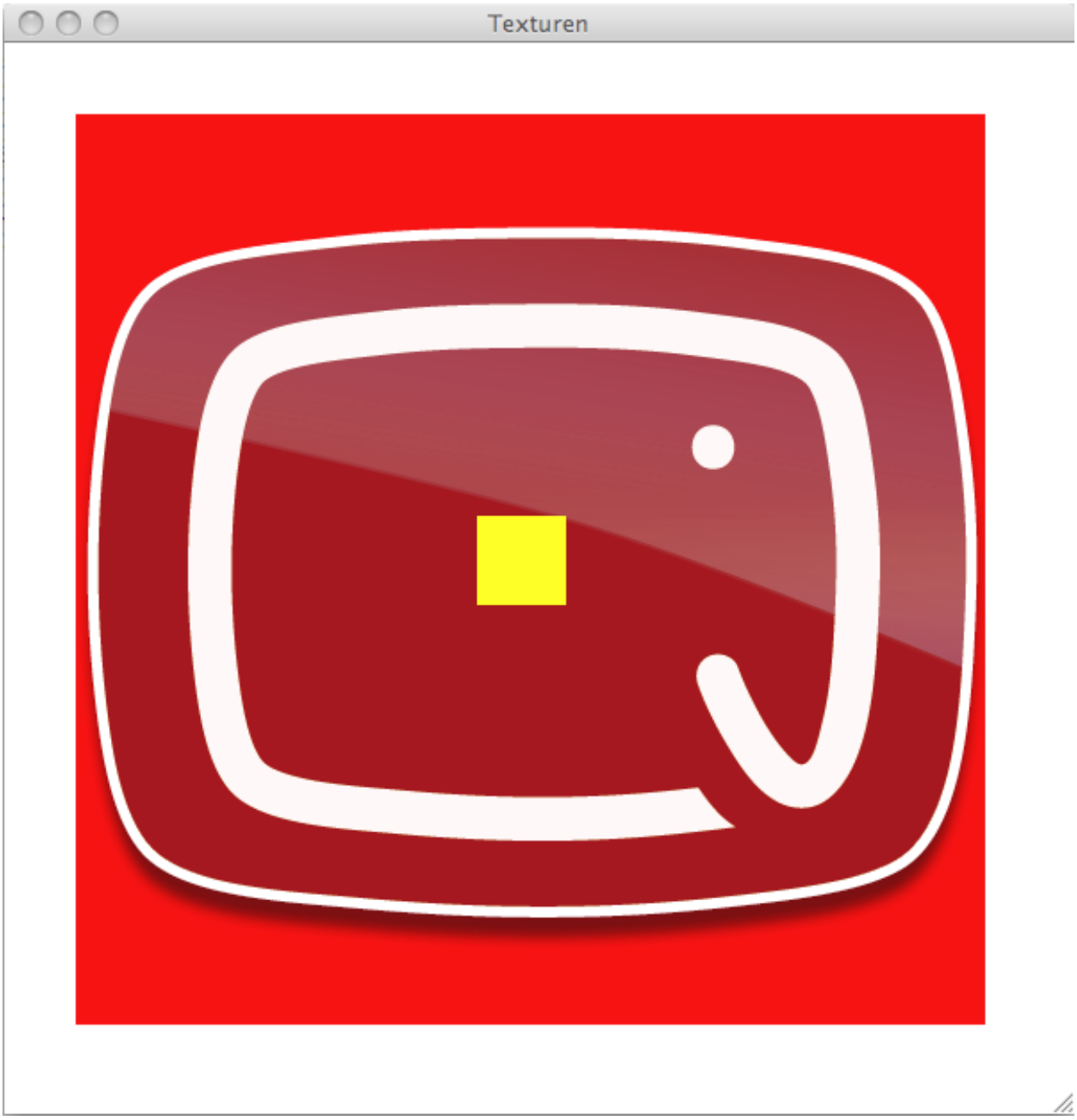
```
<property name="Texture" value="meinbild.png" />
```

Sie können die Bildformate .jpg, .gif und .png verwenden. Grafisch wird die Textur als Hintergrund dargestellt, aber noch vor der Hintergrundfarbe (BackgroundColor) gezeichnet. Texturen dürfen Transparenz enthalten.

Das Textur-Bild wird per Default in Originalgröße oben links im Shape des Widgets angeordnet. Bei freien Formen wird das `Bounding Rect` um die Form errechnet, das dann als Bezugsrahmen genommen wird.

Beispiel:

```
<widget name="widget1">
  <property name="Location" value="80,80"/>
  <property name="Shape" value="R 150 150"/>
  <property name="BackgroundColor" value="red"/>
  <property name="Texture" value="logo.png" />
  <widget name="widget2">
    <property name="Location" value="25,25"/>
    <property name="BackgroundColor" value="yellow"/>
    <property name="Shape" value="R 50 50"/>
  </widget>
</widget>
```



Die Transparenz des eingebundenen PNG-Bildes macht deutlich, dass im Widget zuhinterst die Hintergrundfarbe, dann die Textur gezeichnet werden. Childwidgets sind vor der Textur angeordnet.

Um das Ladeverhalten eines Bildes zu steuern, stehen verschiedene Möglichkeiten zur Verfügung. Vgl. hierzu *Datentypen/Image*.

5.6.2 TextureMode

default: `top-left`

Werte: **`top-left`**, **`top-center`**, **`top-right`**, **`center-left`**, **`center`**, **`center-right`**, `bottom-left`, `bottom-center`, `bottom-right`, `tile`, `force`, `fit`, `cover`, `off`

Mit der Eigenschaft “TextureMode” beeinflussen Sie die Art und Weise, mit der eine Textur in ein Widget eingepasst wird:

```
<property name="TextureMode" type="Symbol" value="center"/>
```

Folgende Werte stehen zur Verfügung:

top-left (Default) Texture-Bild oben links in Originalgröße

ebenso: top-center, top-right, center-left, center, center-right, bottom-left, bottom-center, bottom-right

tile Texture-Bild wird in Originalgröße über die Bounds (= das umschließende Rechteck) des Widgets gekachelt.

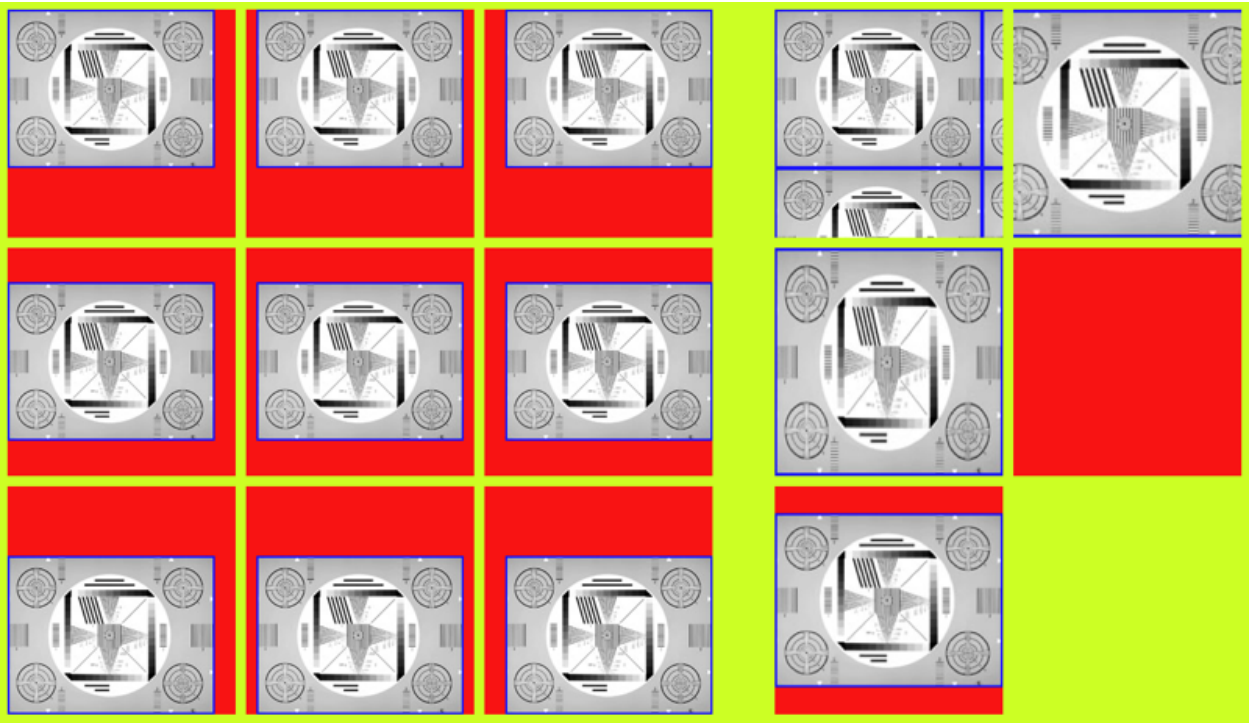
force Das Bild wird in Höhe und Breite auf die Bounds des Widgets skaliert und verzerrt dargestellt, sollten die Seitenverhältnisse von Bounds und Bild nicht übereinstimmen.

fit Das Bild wird in die Bounds des Widgets eingepasst, das Seitenverhältnis wird beibehalten.

cover Das Bild wird so in die Bounds eingepasst, dass das Seitenverhältnis beibehalten wird und die Form vollständig ausgefüllt ist.

off Das Bild wird nicht angezeigt.

Ein Beispiel mit allen möglichen Texturemodes:



Das zugehörige **Mad-Dokument**.

5.6.3 JavaScript

Im folgenden einige beispielhafte Zugriffe und Zuweisungen auf Properties aus diesem Kapitel dargestellt.

Javascript-Syntax:

```
widget1.Texture     = "logo.png";
widget1.Texture     = Type.newImage("logo.png");
widget1.TextureMode = "center";
```

5.7 Interaktion

5.7.1 Interactive (Boolean)

Die Eigenschaft "Interactive" legt fest, dass das Widget Maus-Trigger empfängt:

```
<property name="Interactive" value="true" />
```

Ist die property `false`, so kann durch das Widget "hindurchgeklickt" werden.

Default: `false`

Beim ButtonWidget ist der Default `true`.

Zu den möglichen Triggern vgl. *Actions/Trigger*

5.8 Ausgewählte Methoden der Widget-Klasse

Das Widget stellt zahlreiche Methoden zur Verwendung in Skripten zur Verfügung. Hier werden die am häufigsten genutzten erläutert.

callActionOnNextFrame(String) Diese Methode ruft beim nächsten Frame der Wiedergabe die genannte Action auf.

lookup(String) Mit der Methode `lookup(Suchbegriff)` kann nach Widgets mit dem gesuchten Namen innerhalb des Widgets-Baums gesucht werden. Dabei wird im Namensraum des Widgets selbst und - falls die Bezeichnung dort nicht gefunden wird - vom aufrufenden Widget aufwärts bis in die Presentation gesucht:

```
<widget name="widget1">
  <property name="Location" value="40,40"/>
  <property name="Shape" value="R 50 50"/>
  <property name="BackgroundColor" value="red"/>
</widget>

<widget name="widget2">
  <property name="Location" value="225,225"/>
  <property name="BackgroundColor" value="yellow"/>
  <property name="Shape" value="R 50 50"/>
  <property name="Interactive" value="true"/>
  <action trigger="pointer-down">
   // bei klick auf das Widget wird mittels Lookup eine Referenz
   // auf "widget1" erzeugt auf diesem die Property Location gesetzt.
     this.lookup("widget1").Location = "100,100";
  </action>
</widget>
```

route(Object) Die Methode `route(Object)` versucht das übergebene Objekt anhand der auf dem Widget registrierten Routen weiterzuleiten. Siehe dazu auch das Kapitel zu *Routen*.

queueData(Data) Mittels der Methode `queueData(Data)` wird das übergebene `Data` beim nächsten Framedurchlauf mit dem Trigger `data` als Signal vom Widget in die Anwendung gesendet.

getValueForKey(String) Die Methode `getValueForKey(KeyName)` liefert den für den genannten Key auf dem Widget gespeicherten Wert zurück. Sollte weder eine Property oder eine selbstdefinierte Eigenschaft auf dem Widget existieren, liefert die Methode `null` zurück.

getWithDefault(String, Object) Wie `getValueForKey(Key)`, aber `getWithDefault(Key, DefaultValue)` liefert statt `null` den DefaultValue zurück.

```
<widget name="widget1">
  <property name="Location" value="40,40"/>
  <property name="Interactive" value="true"/>
  <action trigger="pointer-down">
     this.getWithDefault("Name", "Hans")       // liefert "widget1" zurück
     this.getWithDefault("LastName", "Hans")   // liefert "Hans" zurück
  </action>
</widget>
```

addChild(Widget) Mittels `addChild(Widget)` kann ein Widget per Script in den Widget-Baum als Child eingehängt werden.

removeChild(Widget) Diese Funktion entfernt das übergebene Widget aus dem Widget-Baum, sofern es ein Child des aufrufenden Widgets ist. Das entfernte Widget wird zurückgeliefert. Sofern das übergebene Widget nicht gefunden wird, liefert die Methode `null` zurück.

removeChildAt(Number) Entfernt ein Child an einer bestimmten Index-Position und liefert dieses als Rückgabewert zurück.

getChildren() Mittels dieser Funktion kann eine Liste der Children von eines Widgets abgerufen werden. Vgl. auch die Property `Children`.

getChildAt(Number) Hiermit wird ein Child an einer bestimmten Position angesprochen und eine Referenz auf dieses Child zurückgeliefert.

Weitere Methoden des Widgets werden in der API-Dokumentation erläutert.

CLASSDOC:Widget

5.9 Das Type-Value-Prinzip

An vielen Stellen, wo im MAD-XML-Code Werte gesetzt werden müssen, geschieht dies nach dem Type-Value-Prinzip. Der häufigste Anwendungsfall für Type-Value-Angaben ist das Setzen von Widget-Properties. Da eine Widget-Property von einem festgelegten Typ ist, kann die `type` Angabe weggelassen werden. Ausnahmen sind in den Beschreibungen der Widget-Properties dokumentiert.

Elemente, die nach dem Type-Value-Prinzip funktionieren, sind:

- property
- define
- object
- set

Das Zuweisen von Werten kann auf dreierlei Art passieren:

- mit einem `type`- und einem `value`-Attribut
- mit einem `lookup`-Attribut

- mit einem Child-Element, das einen Wert repräsentiert

Zuweisung von Werten mit einem type- und einem value-Attribut

```
<property name="Shape" type="Rectangle" value="200,300" />
<define name="MySmallRect" type="Rectangle" value="200,300" />
<object type="Rectangle" value="200,300" />
<set property="Shape" type="Rectangle" value="200,300" />
```

Zuweisung von Werten mit einem lookup-Attribut

```
<property name="Shape" type="Rectangle" lookup="Path.To.Other.Shape" />
```

Der zu setzende Wert wird an anderer Stelle in der Presentation 'nachgeschlagen'. Die Syntax für Lookups ist im Abschnitt *Defines, Lookups, Bindings* beschrieben.

Die folgende alternative Schreibweise nutzt das `value`-Attribut für ein Lookup und ist gleichbedeutend:

```
<property name="Shape" type="Rectangle" value="@Path.To.Other.Shape" />
```

Zuweisung von Werten als XML-Childnode

```
<property name="Shape">R 200,300</property>
```

oder:

```
<property name="Shape">
  <object type="Shape" value="R 200,300"/>
</property>
```

Im ersten Fall wird die Wertangabe der Property genauso behandelt wie im `value`-Attribut. Es darf kein zusätzliches `value`-Attribut auf dem Property-Element geben.

Im zweiten Fall wird als Wert einer Property ein Objekt vom Typ `Shape` zugewiesen. Neben `<object>` sind hier auch die komplexen Datentypen `<map>`, `<list>`, `<textformat>` und `<data>` erlaubt. `type`- und `value`-Attribute auf dem umschließenden Element sind nicht erlaubt.

Weiteres Beispiel:

```
<define name="Dummymap">
  <map>
    <define name="a" type="String" value="Foo"/>
    <define name="b" type="String" value="Bar"/>
  </map>
</define>
```

Die möglichen Typen und die Formate für die Wertangabe sind im Kapitel *Datentypen* beschrieben.

Zuweisen von dynamischen Ausdrücken

Bei Elementen, die nach dem Type-Value-Prinzip funktionieren, ist auch die Zuweisung von Werten mit einer JavaScript-Funktion im `value`-Attribut oder im XML-Tag möglich. In geschweifte Klammern `{}` wird der Body einer JavaScript-Funktion geschrieben.

Lookups sind innerhalb einer so geschriebenen JavaScript-Funktion nicht möglich, wohl aber der Zugriff auf die Eigenschaften des umschließenden Widgets:

Beispiele:

```
<define name="Test" value="{return 4*4}" />
<define name="Test" value="{return Math.PI * this.Width}" />

alternative Schreibweise:
<define name="Test">{return 4*4}</define>
<define name="Test">{return Math.PI * this.Width}</define>
```

CHAPTER

SIX

MAONGO-TOOLKIT

Das MaongoToolkit versammelt häufig benötigte Widgets, die spezielle Funktionen zur Verfügung stellen. Es gehört zur Standard-Auslieferung von Maongo/MP.

Section author: jo, team

6.1 HTTPEngine

`maongo.core.toolkit.HTTPEngine extends Engine`

Die HTTPEngine dient dazu, aus der Presentation heraus per HTTP-Request Daten anzufordern oder auszutauschen. Dazu kann eine entsprechende Server-`URL` definiert werden.

Die HTTPEngine nutzt die vom Browser bekannte Methode GET.

Bei der HTTPEngine einlaufende Daten aus Serverprozessen können folgende Formate haben:

- XML-Transmissions (Transmission ist eine Sammlung von Data-Objekten; Routing und spezielle Trigger werden unterstützt)
- Data (einzelnes Data-Objekt; Routing und spezielle Trigger werden unterstützt)
- JSON (Objekt oder Liste von Objekten: JSON wird in ein Data umgewandelt, Routing und spezielle Trigger werden unterstützt)
- beliebiges valides XML (kein Routing, generischer Trigger)
- beliebiger Text (kein Routing, generischer Trigger)

Wird ein Data an die HTTPEngine geroutet, so wird es an den Server übertragen. Der Server kann darauf mit einer Transmission antworten, die dann wiederum von der HTTPEngine anhand der in ihr konfigurierten Routen in der Presentation verteilt wird. Eintreffende Transmissions lösen `transmission`- und `data`-Signale aus. Ausgehende Data lösen `request`-Signale aus. Siehe dazu auch *Beispiel: Signale der HTTPEngine*

6.1.1 Properties

URL (URL) Setzt die Serveradresse für die Kommunikation:

```
http://www.maongo.com/path/to/cgi
```

Default: `null`

User (String) Loginname für die Kommunikation mit `URL`.

Default: `null`

Password (String) Passwort für die Kommunikation mit `URL`.

Default: `null`

Encoding (Symbol) Setzt das Text-Encoding für ausgehende Daten.

Default: `"utf-8"`

Werte: `"utf-8"`, `"latin-1"`

Timeout (Time) als Zeitangabe ("30", "30s", "0.5m"). Wird ein HTTP-Request nicht innerhalb der angegebenen Zeit abgeschlossen, so wird er als `communication-failure` behandelt (siehe Signals) und abgebrochen/verworfen.

Default: `30` (30 Sekunden)

Interval (Time) als Zeitangabe. In diesem Zeitabstand wird der Request wiederholt ausgelöst (Polling).

Default: `null` (kein Polling)

ProxyURL (URL): Derzeit ist eine Nutzung dieser Property noch nicht möglich.

Default: `null`

ProxyUser (String): Derzeit ist eine Nutzung dieser Property noch nicht möglich.

Default: `null`

ProxyPassword (String): Derzeit ist eine Nutzung dieser Property noch nicht möglich.

Default: `null`

6.1.2 Signale

Die Signale beziehen sich teilweise auf die Abwicklung der Kommunikation und teilweise auf die geladenen Daten. Je nach geladenem Inhalt können sich die Signale unterscheiden.

communication-request (URL) Die Engine startet einen request

communication-success (Download-Objekt) Die Kommunikation war erfolgreich. Als Argument wird der Action das "rohe" Ergebnis des Requests übertragen. Dabei handelt es sich um ein Download-Objekt. Der Inhalt des Download-Objektes kann im Falle von XML mittels `getSource()` ausgelesen werden.

Beispiel:

```
<action trigger="communication-success" arguments="rawdata">
   trace("rawdata:" + rawdata.getSource());
</action>
```

Mittels `getText()` können die geladenen Daten als Text ausgelesen werden.

Beispiel:

```
<action trigger="communication-success" arguments="rawdata">
   trace("rawdata:" + rawdata.getText());
</action>
```

Bei Erfolg werden außerdem im Falle einer Transmision die spezifischen Signale `transmission-start`, `transmission-end` und `transmission-data` gesendet. Sofern es sich bei den geladenen Daten um XML, aber keine Transmission handelt, wird das Signal `download-xml` gesendet. In allen anderen Fällen wird das Signal `download-text` gesendet.

communication-failure (Exception) Bei der Kommunikation ist ein Fehler aufgetreten.

Spezifische Signale für Transmission und Data

transmission-start Eine Transmission läuft in die Engine ein.

transmission-end (Transmission-Objekt) Eine Transmission ist angekommen; sie wird als Argument übertragen und kann hier direkt genutzt werden:

```
<action trigger="transmission-end" arguments="tr">
        trace(tr.getSerial()); //getSerial() ist eine Methode der Transmission-Klasse.
</action>
```

transmission-data (Data) Ein Data ist angekommen und wird als Argument übertragen:

```
<action trigger="transmission-data" arguments="data">
        trace(data.get("Title"));
</action>
```

Signale für XML und Text

download-xml (RootNode) Ein XML-Dokument ist angekommen. Als Argument wird der RootNode des Dokuments als `SimpleNode` übergeben. Siehe dazu *Datentypen*

download-text (String) Es ist eine undefinierte Art von Daten angekommen. Als Argument werden die empfangenen Daten als String übergeben.

Beispiel: Signale der HTTPEngine

```
<widget name="Box1"  x="10" y="10">
  <property name="BackgroundColor" value="red"/>
  <property name="Interactive" value="true"/>
  <action trigger="pointer-down">
    var commEngine = $(this,"comm");
    commEngine.communicate();
  </action>
</widget>


<engine name="comm" type="Http">
    <property name="URL" value="http://www.maongo.com/examples/bom/resources/example.xml"/>

    <action trigger="communication-request">
        trace("communication-request");
    </action>

    <action trigger="communication-success" arguments="rawdata">
        trace("communication-success: " + rawdata);
    </action>

    <action trigger="communication-failure" arguments="excp">
        trace("communication-failure: " + excp);
    </action>

    <action trigger="transmission-start" arguments="excp">
        trace("transmission-start: " + excp);
    </action>
```

```
    <action trigger="transmission-end" arguments="trans">
        trace("transmission-end: " + trans);
    </action>

    <action trigger="transmission-data" arguments="data">
        trace("transmission-data: " + data);
    </action>

    <action trigger="download-xml" arguments="xmlroot">
        trace("download-xml: " + xmlroot);
    </action>

    <action trigger="download-text" arguments="text">
        trace("download-text: " + text);
    </action>

</engine>
```

6.1.3 Methoden

communicate Die Methode `communicate()` dient dazu, den konfigurierten Request manuell aufzurufen.

Beispiel:

```
$(this, "commEngine").communicate()
```

6.1.4 Routing

Der automatische Routing-Mechanismus für Data weist für Kommunikations-Engines wie die HTTP-Engine eine Besonderheit auf:

Wird ein Data von anderer Stelle in der Presentation auf die Engine geroutet, so wird dieses Data zum Server übertragen.

Note: Das Versenden von Data, welche auf die HTTPEngine geroutet werden ist derzeit nicht implementiert.

Die in der Engine eingetragene Route wird genutzt, um einen vom Server erhaltenen Inhalt zu routen, falls dieser Inhalt geroutet werden kann, also falls er als Data vorliegt.

Aus eingehenden Transmissions werden die einzelnen Data-Objekte so geroutet. Eingehendes JSON wird in ein Data-Objekt verwandelt und ebenfalls geroutet. Jeweils vorausgesetzt, dass entsprechende Routen in der Engine eingetragen sind.

6.1.5 MAD

Beispiel 1:

```
<presentation>

<widget name="Button1" type="Button" x="20" y="20" shape="R 50 50">
  <property name="BackgroundColor" value="olive"/>
  <action trigger="pointer-down">
    $(this, "comm").communicate();
    $(this, "comm").URL = "http://www.maongo.com/notfound";
```

```
  </action>
</widget>

<widget name="Widget2" y="20" x="100" shape="O 50">
  <property name="BackgroundColor" value="gray"/>
</widget>

<engine name="comm" type="Http">
  <property name="URL" value="http://www.maongo.com/examples/bom/resources/example.xml" />
  <route property="*" match="*" target="widget2" />
  <action trigger="transmission-data">
    $(this, "startButton").BackgroundColor = "orange";
  </action>
  <action trigger="communication-success">
    $(this, "Widget2").BackgroundColor = "green";
  </action>
  <action trigger="communication-failure">
    $(this, "Widget2").BackgroundColor = "red";
  </action>
</engine>

</presentation>
```

Dieses Beispiel laden.

Button 1 startet bei Klick die HTTPEngine. Diese färbt bei eingetroffenem Data den Button orange, und je nach Erfolg der Kommunikation mit dem Server das Widget2 grün oder rot. Gleichzeitig wird die URL der Engine auf eine nicht erreichbare Adresse umgeschrieben, sodaß ein zweiter Klick einen communication-failure auslöst.

Beispiel 2:

```
<presentation width="600" height="600">
 <widget type="Button" x="10" y="10">
   <property name="Shape" value="R 150 20"/>
   <property name="Text" value="Transmission laden"/>
   <action trigger="button-clicked">
      $(this, "CommResult").BackgroundColor = "white";
      $(this,"comm").URL = "http://www.maongo.com/examples/bom/resources/basictransmission.xml";
      $(this,"comm").communicate();
   </action>
  </widget>

  <widget type="Button" x="10" y="40">
   <property name="Shape" value="R 150 20"/>
   <property name="Text" value="JSON laden"/>
   <action trigger="button-clicked">
      $(this, "CommResult").BackgroundColor = "white";
      $(this,"comm").URL = "http://www.maongo.com/examples/bom/resources/basicjson.json";
      $(this,"comm").communicate();
   </action>
  </widget>

  <widget type="Button" x="10" y="70">
   <property name="Shape" value="R 150 20"/>
   <property name="Text" value="XML laden"/>
   <action trigger="button-clicked">
      $(this, "CommResult").BackgroundColor = "white";
      $(this,"comm").URL = "http://www.maongo.com/examples/bom/resources/basicxml.xml";
```

```
    $(this,"comm").communicate();
 </action>
</widget>

<widget type="Button" x="10" y="100">
 <property name="Shape" value="R 150 20"/>
 <property name="Text" value="Text laden"/>
 <action trigger="button-clicked">
    $(this, "CommResult").BackgroundColor = "white";
    $(this,"comm").URL = "http://www.maongo.com/examples/bom/resources/basictext.txt";
    $(this,"comm").communicate();
 </action>
</widget>

<widget type="Button" x="10" y="130">
 <property name="Shape" value="R 150 20"/>
 <property name="Text" value="Ladefehler auslösen"/>
 <action trigger="button-clicked">
    $(this, "CommResult").BackgroundColor = "white";
    $(this,"comm").URL = "http://www.maongo.com/examples/bom/resources/nonexisting.xml";
    $(this,"comm").communicate();
 </action>
</widget>

<widget name="CommResult" type="Text" x="300" y="10">
 <property name="Shape" value="R 250 20"/>
 <property name="Text" value="Kommunikation erfolgreich?"/>
 <property name="BackgroundColor" value="white"/>
 <property name="BorderColor" value="black"/>
 <property name="BorderWidth" value="1"/>
</widget>

<engine name="comm" type="Http">
  <property name="URL" value="http://www.maongo.com/examples/bom/resources/example.xml"/>

 <action trigger="communication-request">
    trace("communication-request");
  </action>

 <action trigger="communication-success" arguments="rawdata">
    trace("communication-success: " + rawdata);
    $(this, "CommResult").BackgroundColor = "green";
  </action>

  <action trigger="communication-failure" arguments="excp">
     trace("communication-failure: " + excp);
     $(this, "CommResult").BackgroundColor = "red";
  </action>

  <action trigger="transmission-start" arguments="excp">
    trace("transmission-start: " + excp);
  </action>

  <action trigger="transmission-end" arguments="trans">
    trace("transmission-end: " + trans);
  </action>

  <action trigger="transmission-data" arguments="data">
```

```
      trace("transmission-data: " + data);
    </action>

    <action trigger="download-xml" arguments="xmlroot">
      trace("download-xml: " + xmlroot);
    </action>

    <action trigger="download-text" arguments="text">
      trace("download-text: " + text);
    </action>

  </engine>

</presentation>
```

Dieses Beispiel laden.

In diesem Beispiel können über die verschiedenen Buttons unterschiedliche Datentypen geladen werden. Über die jeweiligen Actions werden die dabei auftretenden Signale ausgegeben. Je nach Erfolg oder Mißerfolg der Kommunikation wird das Widget "CommResult" grün oder rot eingefärbt.

6.2 PekingEngine

`maongo.core.toolkit.PekingEngine extends CommEngine`

Section author: mao

6.2.1 Properties

URL (URL) Setzt die Adresse der state.xml:

```
http://localhost:8080/path/to/state.xml
```

Default: `null`

Timeout (Time) als Zeitangabe ("30", "30s", "0.5m"). Wird ein HTTP-Request nicht innerhalb der angegebenen Zeit abgeschlossen, so wird er als Fehler behandelt.

Default: `120` (120 Sekunden)

Interval (Time) als Zeitangabe. In diesem Zeitabstand wird der Request wiederholt ausgelöst (Polling).

Default: `60` (60 Sekunden)

Lifetime (Time) als Zeitangabe. In diesem Zeitabstand bekommen die Peking-Requests einen neuen Timestamp. Der Wert muss eine Zweierpotenz sein (4,8,16,32,64,128), falls nicht wird die nächsthöhere Zweierpotenz benutzt.

Default: `32` (32 Sekunden)

ProtocolVersion (Integer) Variante des initialen Abrufs der state.xml. Hieraus resultiert auch die Art der Synchronisierung mit der Serverzeit. Wirkt sich derzeit hauptsächlich auf Flash aus

- **1: Default-Zustand: Die Zeitinformation für die Synchronisierung wird aus den Response-Headern ausgelesen und a**
  Der erste Abruf der state.xml wird in Flash mittels eines AJAX-Aufrufs aus der einbindenden HTML-Seite gestartet. Updates bzw. die Channel-Dateien liegen in einem Verzeichnis http://meinserver.de/states/channel-123.xml

- **2: Die Zeitinformation für die Synchronisierung wird aus den Response-Headern ausgelesen und an den Maongo-Cl**
  Der erste Abruf der state.xml wird in Flash mittels eines AJAX-Aufrufs aus der einbindenden HTML-Seite gestartet. Updates bzw. die Channel-Dateien liegen in einem Verzeichnis http://meinserver.de/channel-123.xml
- **3: Die Zeitinformation für die Synchronisierung wird aus der geladenen ersten state.xml ausgelesen.**
  Der erste Abruf der state.xml wird direkt aus Flash gestartet. In der initial abgerufenen state.xml muss ein Attribut ‘time=”123”‘enthalten sein. Updates bzw. die Channel-Dateien liegen in einem Verzeichnis http://meinserver.de/channel-123.xml

Default: `1`

UseFirstXML (Boolean) Sofern diese Property auf `true` gesetzt wird, erfolgt der erste Abruf der `state.xml` nicht auf die Datei `state.xml` sondern auf eine Datei `first.xml`, welche unter dem gleichen Pfad wie `state.xml` erreichbar sein muss. Dieser Aufruf kann für Trackingzwecke genutzt werden, da dieser Aufruf pro aktivem Client nur einmal erfolgt.

Default: `false`

6.2.2 Signals

transmission-start (Transmission)

transmission-data (Data)

transmission-end (Transmission)

6.2.3 Das Peking-Protokoll

Das Peking-Protokoll dient dazu, eine grosse Anzahl von Clients mit einem Datensatz zu versorgen und diese Clientdaten synchron mit dem Originaldatensatz zu halten.

Wichtig sind: Geschwindigkeit: Die Synchronisierung muss schnell passieren. Skalierbarkeit: Das Protokoll kann durch übliche Lastverteilungen hindurch (Akamai) viele zehntausend Clients versorgen. Zuverlässigkeit: Alle Clients bleiben synchron mit dem Originaldatensatz

Ein Datensatz heisst Channel. Er beinhaltet ein oder mehrere Data-Elemente, die im Rahmen des PP versioniert werden.

Nach einer ersten Versorgung eines Clients mit einem Basisdatensatz (“channel-123.xml”) lädt dieser weitere Daten noch inkrementell als Updates nach (“update-124.xml”, “update-125.xml”, ...)

Der aktuelle Zustand eines Channels heisst “state” und wird in einer Datei “state.xml” verwaltet.

Ablauf einer Client-Channel Kommunikation

1) Kontaktaufnahme Der Client lädt die state.xml so, dass jeder Cache umgangen wird, üblicherweise mit einem Zufalls-Parameter. http://my.server.com/mychannel/state.xml?837645283648724628236 4 Der Client wertet das von Server zurückgegebene “Date” Field im http-header aus um einen Offset zwischen der eigenen Zeit und der Serverzeit zu errechnen.

2) Basisdaten In der state.xml findet der Client die aktuelle serial-number des Channels (z.B.42). Er kann den entsprechenden Basisdatensatz jetzt aus der datei “channel-42.xml” laden.

3) Polling In regelmässigen Abständen pollt der Client die state.xml. Er verwendet dabei eine modifizierte Serverzeit als parameter. Diese errechnet sich wie folgt:

Die Serverzeit in sekunden (seit 1970) wird errechnet aus eigener Zeit und Serveroffset. Die letzten 6 (5/7) bits werden auf 0 gesetzt, so dass sich der Parameter alle 32 (16/64) sekunden ändert.

Wird eine neue serial-number im state.xml gefunden, so muss der Client jedes Update zwischen seiner aktuellen lokalen version und der im state.xml angegebenen serial laden.

Peking II

Neu: Pro Channel ein Verzeichnis, alle Dateien liegen darin.

state.xml - wie bisher, aber jetzt im Verzeichnis channel.xml - eine kopie der aktuellsten channel.xml

channel-123.xml update-123.xml

Inhalt der state.xml

Minimal: <state serial="123" protocol="2"/>

Angabe der lifetime als zweierpotenz: <state serial="123" protocol="2" lt="5"/>

Angabe des Poll-Interval in sekunden <state serial="123" protocol="2" pi="30"/>

6.3 SocketEngine

`maongo.core.toolkit.SocketEngine extends Engine`

Section author: jo

Note: XXXXXXXXXXXXXXX Status 01.11.2010: Zur Diskussion XXXXXXXXXXXXXXXXXXXX

Die SocketEngine dient dazu, aus der Presentation heraus eine Serververbindung zu öffnen (TCP-Socket zu `Server``auf ``Port`), über die beidseitig Data ausgetauscht werden können.

Bei der SocketEngine einlaufende Daten aus Serverprozessen können folgende Formate haben:

- XML-Transmissions (Transmission ist eine Sammlung von Data-Objekten; Routing und spezielle Trigger werden unterstützt)
- Data (einzelnes Data-Objekt; Routing und spezielle Trigger werden unterstützt) *?*
- JSON (Objekt oder Liste von Objekten: JSON wird in ein Data umgewandelt, Routing und spezielle Trigger werden unterstützt)
- beliebiges valides XML (kein Routing, generischer Trigger)

Zu übertragende Daten (Data) müssen auf die SocketEngine geroutet werden. Der Server kann die TCP-Connection ebenfalls nutzen, um Daten zu schicken, die dann ggf. in Data umgewandelt werden und anhand der in der Engine konfigurierten Routen in der Presentation verteilt wird. Eintreffende Transmissions lösen `transmission`- und `data`-Signale aus. Ausgehende Data lösen `transmit`-Signale aus. *?*

6.3.1 Properties

Server (URL) Setzt die Serveradresse für die Kommunikation:

```
server.com
172.16.0.123
localhost
```

Default: null

Port (Integer) Nummer des TCP-Ports, über den kommuniziert werden soll.

Default: null

Password (String) ***?*** Passwort für die Kommunikation mit `Server`.

Default: null

Encoding (Symbol) Setzt das Text-Encoding für ausgehende Daten.

Default: `"utf-8"`

Werte: `"utf-8"`, `"latin-1"`

PingInterval (Time) *NICHT Interval*

als Zeitangabe.

Default: 30 (30 Sekunden)

6.3.2 Signale

communication-start **(Data)** Ein Data ist auf die SocketEngine geroutet worden und wird zum Server weitergeleitet.

communication-success ***kann es das geben?*** Bei der Kommunikation ist ein Fehler aufgetreten.

communication-failure (Params?) Bei der Kommunikation ist ein Fehler aufgetreten.

communication-connect Eine dauerhafte Serververbindung wurde gestartet

communication-disconnect Eine dauerhafte Serververbindung wurde unterbrochen

communication-ping Der Server hat das gesendete Ping bestätigt.

send-start Start der Datenübertragung ZUM Server. (Socket und HTTP)

send-end Ende der Datenübertragung ZUM Server. (Socket und HTTP)

receive-start Start der Datenübertragung VOM Server. (Socket und HTTP)

receive-end Ende der Datenübertragung VOM Server. (Socket und HTTP)

Transmission und Data:

transmission (Transmission-Objekt) Eine Transmission ist angekommen; sie wird als Argument übertragen und kann hier direkt genutzt werden:

```
<action trigger="transmission" arguments="tr">
        trace(tr.getSerial()); //getSerial() ist eine Methode der Transmission-Klasse.
</action>
```

data (Data) Ein Data ist angekommen und wird als Argument übertragen:

```
<action trigger="data" arguments="data">
        trace(data.get("Title"));
</action>
```

6.3.3 Methoden

communicate Führt die in der Engine definierte Kommunikation aus.

Beispiel:

```
$(this, "commWidget").communicate()
```

6.3.4 Routing

Der automatische Routing-Mechanismus für Data weist für Kommunikations-Engines wie die SocketEngine eine Besonderheit auf:

Wird ein Data von anderer Stelle in der Presentation auf die Engine geroutet, so wird dieses Data zum Server übertragen. Die in der Engine eingetragene Route wird genutzt, um einen vom Server erhaltenen Inhalt zu routen, falls dieser Inhalt geroutet werden kann, also falls er als Data vorliegt.

Aus eingehenden Transmissions werden die einzelnen Data-Objekte so geroutet. Eingehendes JSON wird in ein Data-Objekt verwandelt und ebenfalls geroutet. Jeweils vorausgesetzt, dass entsprechende Routen in der Engine eingetragen sind.

Section author: jo

6.4 AudioEngine

Section author: jo

Note: unvollständig

Audio-Engine: Grundidee ist, sich an der HTML5-Spezifikation zu orientieren http://dev.w3.org/html5/spec/Overview.html#audio

Noch mal nachdenken über AudioWidget: das controls-Attribut ist schon cool...

Achtung: Das ist obsolet. AudioEngine evtl. nur noch, um Eventsounds abzuspielen.

Engine allgemein

Während Widgets in einer Presentation stets als **sichtbare** Elemente auftauchen, sind Engines unsichtbare Funktionselemente der Presentation.

Sie werden im MAD-XML-Code an beliebiger Stelle definiert und können dann - wie Widgets - als Routingziele, für Lookups etc. verwendet werden. Engines werden - ebenfalls analog zu Widgets - mit Properties konfiguriert.

Es ist nicht erlaubt, innerhalb eines <engine>-Tags Widgets zu definieren.

AudioEngine jetzt aber

Erlaubt das Abspielen von Audio-Dateien und -Streams.

Unterstützte Formate:

- MP3-Dateien über (HTTP- oder File-) URLs (audio/mpeg)
- MP3-Livestreams über HTTP-URLs (audio/mpeg)
- PLS-Dateien über (HTTP- oder File-) URLs, die MP3-Dateien oder -Livestreams referenzieren

```
<engine name="MyAudioPlayer" type="Audio">
        <property name="Source" value="[lazy http://www.domain.com/einfile.mp3]" />
        <property name="AutoPlay" value="false" />
        <property name="Loop" value="0" />
        <property name="Control" value="pause" />
</engine>
```

6.4.1 Properties

Source –analog HTML5 (src); HTML5 has multiple sources: <source> as child of <audio>

AutoPlay –analog HTML5

Loop –analog HTML5

Control

AutoBuffer – defaults to true <–> conflicts with Source Syntax ?!

CurrentTime –analog HTML5/JS

6.4.2 Signale

audio-loadstart oder media-

audio-load

audio-start

audio-pause

audio-end

audio-frame

6.4.3 Methoden

```
getFrame()

getTimecode()

?getBufferSize()

?setTime()

?setFrame()

?setTimeCode()

canPlayType() -- analog HTML5
```

6.4.4 JavaScript

```
MyAudioPlayer.Control = "play";
```

6.5 ButtonWidget

`maongo.core.toolkit.ButtonWidget extends Widget`

Note: ########## gegenlesen laeuft / 24.3.11 ####################

Ein einfacher **Button** führt eine Aktion bei Klick aus:

```
Beispiel 1:
<?xml version="1.0" encoding="UTF-8"?>
<presentation width="340" height="200">

  <!-- erster Button -->
  <widget name="TheButton1" type="Button" x="20" y="20" shape="R 50 50">
  <action trigger="button-clicked">
     this.BackgroundColor = "red";
  </action>
  </widget>

  <!-- zweiter Button -->
  <widget name="TheButton2" type="Button" x="90" y="20" shape="R 50 50">
  <action trigger="button-clicked">
    this.BackgroundColor = "yellow";
  </action>
  </widget>

  <!-- dritter Button -->
  <widget name="TheButton3" type="Button" x="160" y="20" shape="R 50 50">
  <action trigger="button-clicked">
    $(this, "TheButton1").BackgroundColor = "white";
    $(this, "TheButton2").BackgroundColor = "white";
    this.BackgroundColor = "green";
  </action>
  </widget>

</presentation>
```

Bei Klick auf den ersten Button wird die Hintergrundfarbe von `TheButton1` auf rot gesetzt, bei Klick auf den zweiten Button wird die Hintergrundfarbe von `TheButton2` auf gelb gesetzt, und ein Klick auf den dritten Button setzt die Farbe der ersten beiden Buttons auf weiß und sich selber auf grün.

Ein **Data-Button** speichert ein ankommendes Data und sendet es bei Klick weiter:

```
Beispiel 2:

<!-- Data-Transmission -->
<transmission id="1">
  <data action="Update">
    <property name="myColor" type="Color">#ffff00</property>
  </data>
</transmission>

<!-- Presentation -->
<presentation>
  <route property="*" match="*" target="btn" />
  <widget name="btn" type="Button" location="10,10">
    <property name="DataMode" value="store-forward" />
    <route property="*" match="*" target="aWidget" />
  </widget>
  <widget name="aWidget" location="120,10">
    <bind property="BackgroundColor" to="DataObserver" key="myColor" />
    <property name="BorderWidth" value="1" />
  </widget>
</presentation>
```

Das bei Aufruf der Presentation geladene Data wird an den Button geroutet und dort gespeichert, bis ein Klick erfolgt. Dann wird es anhand der in `btn` angegebenen Route weitergeleitet zu `aWidget`, wo über ein Binding eine Property aus dem Data ausgelesen wird. `aWidget` wird die in `myColor` gespeicherte Farbe annehmen. Näheres zu Data und Binding siehe *Data und Routen*.

Eine **Buttongruppe** erlaubt das Umschalten zwischen verschiedenen Optionen:

```
Beispiel 3:

<presentation width="340" height="200">
  <widget name="btn1" type="Button" x="20" y="50" shape="R 20 20">
    <property name="Mode" value="radio" />
    <property name="Group" value="myButtons" />
    <property name="State" value="on" />
  </widget>
  <widget name="btn2" type="Button" x="50" y="50" shape="R 20 20">
    <property name="Mode" value="radio" />
    <property name="Group" value="myButtons" />
  </widget>
  <widget name="btn3" type="Button" x="80" y="50" shape="R 20 20">
    <property name="Mode" value="radio" />
    <property name="Group" value="myButtons" />
  </widget>
  <widget name="btn4" type="Button" x="110" y="50" shape="R 20 20">
    <action trigger="button-clicked">
    <![CDATA[
        // NYI: trace($(this, "Presentation").getButtonGroup('myButtons').getPressed());
        // see codebase #256 for bugs
        trace("Button 1:" + $(this, "btn1").State);
        trace("Button 2:" + $(this, "btn2").State);
        trace("Button 3:" + $(this, "btn3").State);
    ]]>
    </action>
  </widget>

</presentation>
```

Beim Start der Presentation wird `btn1` eingeschaltet (State = *on*). Ein Klick auf `btn2` oder `btn3` wird `btn1` ausschalten und den geklickten Button einschalten. `btn4` gehört nicht zur ButtonGroup. Klick auf `btn4` gibt hier den aktuell gedrückten Button der ButtonGroup `myButtons` aus.

6.5.1 Funktionsweisen des ButtonWidgets

Das Button-Widget kennt drei Verhaltensweisen, die über die Property `ButtonType` eingestellt werden.

Die folgende Aufstellung gibt an, welche Zustandsänderungen bei zwei vollständigen Klicks auf den Button (Press und Release auf dem ButtonWidget) stattfinden und wann jeweils die Buttonaktionen ausgeführt werden.

Verhalten bei...

... ButtonType = `"click"`:

```
Aktion      State   Signal            Aktion?

Press       'on'    button-pressed
Release     'off'   button-released   ButtonAktion
Press       'on'    button-pressed
Release     'off'   button-released   ButtonAktion
```

... ButtonType = "toggle":

```
Press       'on'    button-pressed
Release     'on'                    ButtonAktion
Press       'on'
Release     'off'   button-released
```

... ButtonType = "radio":

```
Press       'on'    button-pressed
Release     'on'                    ButtonAktion
Press       'on'
Release     'on'

***Press*** 'off'   button-released
(wird released bei Klick auf einen anderen Button der Gruppe)
```

Das direkte Setzen der Property ButtonState auf "on" bzw. "off" ist möglich, um eigene Buttonfunktionen zu implementieren. Eine solche Property-Änderung löst direkt button-pressed bzw. button-released aus, unabhängig vom ButtonType.

Mit der Property ActionOnPress lässt sich außerdem bewirken, dass die Buttonaktion bereits bei der Press-Aktion ausgeführt wird.

Button und Data:

Der Button verhält sich in Bezug auf Data, die durch die Anwendung geroutet werden, standardmäßig wie ein normales Widget: Es leitet das Data weiter, wenn entsprechende Routen auf dem Widget definiert sind.

Zusätzlich speichert der Button das Data in seiner Property Data (was ein data-Signal auslöst). Das ButtonWidget registriert sich außerdem als Listener für Änderungen im Data und sendet ein data-changed-Signal, wenn sich das gespeicherte Data ändert.

Die Property DataMode ermöglicht es, dieses Verhalten zu modifizieren. Standardwert ist "route", mit der genau das geschilderte Verhalten erreicht wird.

Verhalten bei...

... DataMode = "store-forward": Das geroutete Data wird wie beschrieben gespeichert, und erst bei Klick wird das Data anhand der Routen weiter geroutet.

... DataMode = "request": Das geroutete Data wird wie beschrieben gespeichert, bei Klick wird ein Request gestartet, um eine Transmission von der in der Property DataSource definierten URL abzuholen. Bei Eintreffen der Data-Transmission werden die enthaltenen Data anhand der Routen des ButtonWidgets weiter geroutet.

Ein Data kann auch direkt der Property Data auf dem ButtonWidget zugewiesen werden. Ist dies der Fall, so wird das zugewiesene Data nicht automatisch weiter geroutet. Es ersetzt aber das gespeicherte Data und würde im DataMode store-forward bei Klick weiter geroutet.

LabelTemplates:

Der Button nutzt die Property Text, um eine einfache Buttonbeschriftung anzuzeigen.

Soll die Buttonbeschriftung spezifisch formatiert sein, so können Sie beliebige weitere Widgets in den Button legen, die die Darstellung übernehmen. Binden Sie an die Property Text des Buttons, um den Wert der Property an anderer Stelle zu verwenden.:

```
Beispiel 4
<presentation width="230" height="230">

  <widget name="mybutton" type="Button" x="30" y="30">
```

```
      <property name="Text" value="Button-Text" />

      <!-- dieses Textwidget wird als Label verwendet -->
      <widget type="Text">
        <property name="ForegroundColor" value="orange" />
        <property name="TextAlign" value="right" />
        <bind property="Text" to="parent" key="Text" />
       </widget>

    </widget>

</presentation>
```

Der Button `mybutton` nutzt ein vorformatiertes Textwidget; die Property `Text` des Childwidgets ist an die Property `Text` des Parents gebunden.

6.5.2 Properties

Das ButtonWidget hat für zwei von Widget ererbte Properties andere Defaultwerte als das Widget:

`BorderWidth`: Default `1.0` (Widget: `0.0`)

`Interactive`: Default `true` (Widget: `false`)

Eigene Properties:

ActionOnPress (Boolean) Per Default werden Actions beim Loslassen (`button-released`) ausgeführt. Diese Property erlaubt es, dieses Verhalten zu modifizieren, um Aktionen bereits auf Drücken (`button-pressed`) auszuführen.

Default: `false`

ButtonGroup (String) Erlaubt die Zuordnung eines Buttons zu einer Gruppe. Ist der Typ der gruppierten Buttons `radio`, so werden beim Einschalten eines Buttons alle anderen ausgeschaltet (wird der ButtonState auf `0` gesetzt). Ist der ButtonType `toggle` oder `click`, so hat die ButtonGroup keinen Effekt.

Gruppen gelten innerhalb einer Presentation.

Default: `""` (keine ButtonGroup)

ButtonState (Symbol) Setzt den Zustand des Buttons auf `on` oder `off`. Wenn damit eine State-Änderung einhergeht (d.h., wenn der Button nicht schon im gesetzten Zustand war), wird ein `button-pressed`- oder ein `button-released`-Signal gesendet.

Default: `"off"`

ButtonType (Symbol) Legt die Verhaltensweise des Buttons fest. Mögliche Werte sind: `click`, `toggle` und `radio`.

Default: `"click"`

Data (Object) [WidgetProperty] Die Property erlaubt es, dem Button ein Data zuzuweisen (durch direktes Setzen der Property in einem JavaScript oder über Binding).

Wird die Property `Data` auf ein Object vom Typ `Data` gesetzt, so sendet der Button ein `data`-Signal. Ändert sich das gespeicherte Data, so sendet der Button ein `data-changed`-Signal.

Ein auf dem Button gespeichertes Data wird erst ersetzt, wenn ein neues Data einläuft oder gesetzt wird.

Default: `null` (kein Data)

Beispiel:

```
<presentation width="230" height="230">

 <widget name="mybutton" type="Button" x="30" y="30">
   <property name="Shape" value="R 80,20"/>
   <property name="Text" value="Button-Text" />
   <action trigger="data" args="d">
     trace("Data erhalten" + d);
   </action>
 </widget>


 <widget name="clickbtn" type="Button" x="30" y="50">
   <property name="Shape" value="R 80,20"/>
   <property name="Text" value="set data" />
   <action trigger="button-clicked">
     var d = Type.newData("Hans");
     $(this,"mybutton").Data = d;
   </action>
 </widget>
</presentation>
```

DataMode (Symbol) Der DataMode `store-forward` legt fest, dass der Button ein auf ihn geroutetes Data automatisch in die Property Data speichert und erst bei Klick weiterleitet (entlang der gesetzten Routen). Der DataMode `request` legt fest, dass DataSource genutzt wird, um bei Klick eine Transmission anzufordern.

Steht DataMode auf dem Defaultwert `"route"`, so verhält sich das ButtonWidget wie ein normales Widget und leitet Data, die es empfängt, anhand seiner Routen direkt weiter.

Werte: `"route"`, `"store-forward"`, `"request"`

Default: `"route"` (normales Widget-Verhalten, kein Data-Button)

DataSource (URL) Wenn der DataMode des Buttons `request` ist, wird bei Klick auf den Button eine Transmission von der in `DataSource` angegebenen URL geladen und das erste Data der Property Data zugewiesen. Enthält die Transmission weitere Data, so werden diese verworfen. Ist DataMode `request` und DataSource `null`, so wird nichts geladen.

Default: `null` (keine DataSource)

Text (String) Die Property erlaubt es, eine Buttonbeschriftung anzugeben.

Default: `""` (Keine Beschriftung)

6.5.3 Methoden

click simuliert einen Klick auf den Button (was die Ausführung eines Klicks aus JavaScript heraus ermöglicht).

press simuliert einen Drücken auf dem Button.

release simuliert ein Loslassen auf dem Button.

Beispiel:

```
$(this, "aButton").click()
```

6.5.4 Signale

button-pressed Signal, das anzeigt, dass der Button gedrückt wurde.

button-released Signal, das anzeigt, dass der gedrückte Button losgelassen wurde. Das Release ist auf dem Button selbst erfolgt.

button-released-outside Signal, das anzeigt, dass der gedrückte Button losgelassen wurde. Das Release ist nicht auf dem Button erfolgt.

button-clicked Bei Mausbedienung das Signal, das einen vollständigen Klick (Press und Release innerhalb des Shape des Buttons) anzeigt.

Vgl. auch *Actions* und *Trigger*.

6.6 TextWidget

`maongo.core.toolkit.TextWidget extends Widget`

Note: Noch nicht implementiert.

Das TextWidget stellt einzeiligen oder mehrzeiligen Text dar. Bei mehrzeiligem Text erfolgen automatische Textumbrüche wortweise.

Soll der Text vom Rand des Widgets abgerückt werden, kann dafür die Widget-Property `Padding` gesetzt werden.

Im Widget ist nicht unbedingt der gesamte Text sichtbar. Durch Widget-`Shape`, -`Padding`, aber auch `TextAlign`, `HorizontalScrollPosition` und `VerticalScrollPosition` wird der sichtbare Bereich auf den Text verändert.

Der Inhalt der `Text`-Property kann mit unterschiedlichen Formattern formatiert werden: `plaintext` (Default) stellt den Text direkt dar; `markup` nutzt eine einfache Markup-Syntax, um Zeilenumbrüche, Absatzformate u.ä. zuzuweisen; `printf` nutzt die `Text`-Property als Formatierungsstring, in den ein oder mehrere Werte der `Value`-Property eingefügt werden. Weitere Formatierungsoptionen s.u..

6.6.1 Properties

MultiLine (Boolean) Wenn `true` wird der Text mehrzeilig dargestellt und dafür wortweise umgebrochen. Wenn `false` ist die Textdarstellung einzeilig.

Default: `false`

Text (String) Der darzustellende Text.

Default: `""`

TextAlign (Symbol) Die Ausrichtung des gesamten Textes innerhalb des Widgets (horizontal und vertikal).

Es wird der gesamte Textblock an die Position bewegt.

Werte:

```
top-left      top      top-right
left          center   right
bottom-left   bottom   bottom-right
```

Default: `top-left`

TextFormat (TextFormat) (*Nicht alle Properties von TextFormat funtionieren, siehe TextFormat*) Das Default-Textformat.

Default: eine Default-Textformat (Default-Werte aller Properties des TextFormat-Objekts)

Werte: TextFormat-Objekte. `<textformat>`-Objekte können an beliebiger Stelle im XML definiert werden, auch direkt im Property-Tag.

TextStyles (SimpleMap) Eine Sammlung von Textformaten zur Darstellung von ausgezeichnetem Text.

Default: `null` (keine zugewiesenen Styles)

Werte: Styles-Objekte. Styles sind Sammlungen von TextFormat-Objekten. `<styles>` können an beliebiger Stelle im XML definiert werden, auch direkt im Property-Tag.

HorizontalScrollPosition (Number) Ein Offset des Textblocks in horizontaler Richtung. Positive Werte bezeichnen die Verschiebung des Textblocks nach rechts. 0 ist keine Verschiebung, negative Werte bezeichnen eine Verschiebung nach links. (**???**)

Dient zur Implementierung eines einfachen Scrollverhaltens in Actions.

Default: `0`

VerticalScrollPosition (Number) Ein Offset des Textblocks in vertikaler Richtung. Positive Werte bezeichnen die Verschiebung des Textblocks nach unten. 0 ist keine Verschiebung, negative Werte bezeichnen die Verschiebung nach oben. (**???**)

Dient zur Implementierung eines einfachen Scrollverhaltens in Actions.

Default: `0`

Formatter (Symbol) (NYI) Ein Formatierer, der die 'Value' property formatiert und darstellt. Je nach ausgewähltem Formatter wird die `Text` Property unterschiedlich genutzt

Default: `plaintext`

Werte: `plaintext`, `markup`, `printf` sowie verschiedene Shortcuts für Zahlen- und Datumsformate, vgl. *Formatter*.

Value (Object) (NYI) Ein oder mehrere Werte für den Formatter

Default: `null` (kein Wert gesetzt)

Werte: Alle einfachen Types oder eine SimpleMap (SimpleMap nur für Formatter `printf` nutzbar).

FixedWidth (Number) (NYI) (NYI in Actionscript) Breitenvorgabe für den Text: Soweit nicht ein Textformat eine andere Vorgabe macht, wird der Text auf diese Breite umgebrochen. An Layoutmanager wird dieser Wert (plus Padding) als Breite der PreferredBounds übergeben.

Default: `0` (nicht gesetzt)

FixedLines (**KILL**)

> **Warning:** Je länger ich darüber nachdenke umso weniger Sinn scheint mir das zu machen.
> Ein TextLayouter, der mit einer festen zeilenzahl arbeitet wäre ganz neu zu schreiben, ob der dann funktioniert ist fraglich.
> Ich glaube, wir sollten das erstmal weglassen.
> -mao-
> Höhenvorgabe für den Text: In Layouts wird dieser Wert (als FixedLines*(LineHeight + LineSpacing)) als Höhe der PreferredBounds an den Layoutmanager übergeben. Es werden maximal n Zeilen dargestellt, auch wenn das Widget mehr Platz bietet.
> Default: `null` (nicht gesetzt)

MaxLines (Integer) (NYI) (NYI in Actionscript) Vorgabe: Es werden maximal n Zeilen Text dargestellt, auch wenn das Widget mehr Platz bietet. Hat keinen Einfluss auf Layouts.

Default: `0` (nicht gesetzt)

`FixedWidth` und `FixedLines` oder `FixedWidth` und `MaxLines` beschreiben bereits vollständig die Dimensionen, die für die Textdarstellung genutzt werden. Sind `FixedLines` und `MaxLines` gleichzeitig gesetzt, so gilt der kleinere Wert. (?)

6.6.2 Methoden

keine

6.6.3 Signale

keine

6.6.4 TextFormat

TextFormat-Objekte enthalten informationen zur Formatierung von Textinhalten. Sie können (wie <define>, <map>, etc.) in einem Widget definiert werden oder direkt als Wert an die Property `TextFormat` des TextWidgets übergeben werden.

Syntax-Beispiel:

```
<!-- geht so nicht -->
<presentation>

        <typeface name="MyFont" source="Arial.ttf" autostyles="true" />

        <typeface name="MyFont2" source="Arial.ttf">
                <font name="italic" bold="false" italic="true"/>
                <font name="bold" bold="true" italic="false"/>
                <font name="bold-italic" bold="true" italic="true"/>
        </typeface>

        <typeface name="SpecialFont" source="Special.ttf">
                <font name="italic" source="Special-Regular-Kursiv.ttf" />
                <font name="bold"  source="Special-Condensed-Strong.ttf" />
                <font name="bold-italic"  source="Special-Condensed-Kursiv.ttf" />
                <font name="boldest"  source="Special-ExtraBlack.ttf" />
        </typeface>

        <textformat name="aFormat">
                <property name="FontFamily" value="MyFont"/>
                <property name="FontSize" value="12.2"/>
                <property name="BaselineOffset" value="2"/>
        </textformat>

        <textformat name="bFormat" extends="aFormat">
                <property name="FontColor" value="#ff0000"/>
        </textformat>

        <widget name="widget1" type="Text">
                <property name="TextFormat" lookup="bFormat"/>
        </widget>

        <widget name="widget2" type="Text">
                <property name="TextFormat">
```

```
                            <textformat extends="aFormat">
                                    <property name="FontColor" value="#0000ff"/>
                            </textformat>
                        </property>
                </widget>

        </presentation>

<!-- das geht: -->
<?xml version="1.0" encoding="UTF-8"?>
<presentation xmlns="http://www.example.org/mad"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/mad mad.xsd "
  width="340" height="200">

  <typeface name="FrutigerNext">
    <font style="roman" name="FrutigerNext-Roman" source="fonts/SWBTR___.TTF" />
    <font style="bold" name="FrutigerNext-Bold" source="fonts/SWBTB2__.TTF" />
    <font style="heavy" name="FrutigerNext-Heavy" source="fonts/SWBTH___.TTF" />
  </typeface>
  <typeface name="ZDFnewsScreen">
    <font style="roman" name="FrutigerNext-Roman" source="fonts/SWBTR___.TTF" />
    <font style="bold" name="FrutigerNext-Bold" source="fonts/SWBTB2__.TTF" />
    <font style="heavy" name="FrutigerNext-Heavy" source="fonts/SWBTH___.TTF" />
  </typeface>
  <typeface name="MyFont2">
    <font name="italic" source="Arial.ttf"/>
    <font name="bold" source="Arial.ttf"/>
    <font name="bold-italic" source="Arial.ttf"/>
  </typeface>


  <textformat name="Roman16">
    <property name="FontFamily" value="FrutigerNext"/>
    <property name="FontSize" type="Number" value="16"/>
    <property name="FontColor" type="Color" value="#2e324d"/>
    <property name="FontStyle" value="roman" />
  </textformat>

  <textformat name="Bold">
    <property name="FontFamily" value="ZDFnewsScreen"/>
    <property name="FontSize" type="Number" value="16"/>
    <property name="FontColor" type="Color" value="#ffffff"/>
    <property name="FontStyle" value="bold" />
  </textformat>

  <textformat name="aFormat">
      <property name="FontFamily" value="MyFont2"/>
      <property name="FontSize" type="Number" value="12.2"/>
      <property name="BaselineOffset" type="Number" value="2"/>
      <property name="FontColor" type="Color" value="#000000"/>
  </textformat>

  <textformat name="bFormat" extends="aFormat">
      <property name="FontColor" type="Color" value="#ff0000"/>
  </textformat>

  <widget name="widget1" type="Text" x="10" y="10">
```

```
      <property name="TextFormat" lookup="bFormat"/>
      <property name="Text" value="Ich bin Widget 1!" />
  </widget>

  <widget name="widget2" type="Text" x="120" y="10">
    <property name="TextFormat">
      <textformat extends="aFormat">
          <property name="FontColor" type="Color" value="#0000ff"/>
      </textformat>
    </property>
    <property name="Text" value="Ich bin Widget 2!" />
  </widget>

</presentation>
```

Das Beispiel präsentiert zunächst drei Optionen, Fonts zu definieren. Im Font-Tag wird das (Default-)Source-File des Fonts angegeben, das (mindestens) den Style `plain` definiert. Die Angabe `autostyles == true` bewirkt, dass außerdem die Styles `bold`, `italic` und `bold-italic` zur Verfügung stehen (falls der Font diese Schriftschnitte enthält und standardgemäß verwaltet). Die Deklarationen von `MyFont` und MyFont2‘‘ haben daher dieselbe Auswirkung.

Wird `autostyles` nicht oder auf `false` gesetzt, so steht nur der `plain`-Schnitt automatisch zur Verfügung.

`MyFont2` nutzt eine Fontdatei, die mindestens die vier Standardschnitte enthält; nach dieser Definition sind die Styles `plain`, `italic`, `bold` und `bold-italic` nutzbar.

`SpecialFont` definiert alle FontStyles mit eigenen `source`-Angaben. Hier ist auch gezeigt, wie ein eigener Schriftschnitt (`boldest`) zusätzlich angelegt werden kann. fehlen die Attribute `bold` und *ìtalic‘* im `<font>`-Tag, so werden sie als `false` angenommen.

Das Widget `widget1` nutzt das Textformat `bFormat`, das eine Erweiterung des Textformats `aFormat` ist. (Näheres zur Vererbung bei Textformaten siehe *Datentypen*). Auf diese Weise werden hier `FontFamily`, `FontSize`, `BaselineOffset` aus `aFormat` und `FontColor` aus `bFormat` zugewiesen.

Alternativ kann das Textformat auch direkt als XML-Wert definiert werden wie bei `widget2`.

Properties von TextFormat

Textformat-Objekte können zur Laufzeit nicht modifiziert werden. Die folgenden Properties können daher nur gelesen, nicht aber gesetzt werden.

Primäre Properties zur Schriftformatierung

FontFamily (String) FontFamily-Angabe (Name eine im XML definierten <fontfamily> oder einer eingebauten FontFamily.

Default: `sans` (eingebaute FontFamily)

Werte: `sans` oder eigene FontFamily.

FontSize (Number) Schriftgröße in Pixel

Default: `12`

FontStyle (Symbol)

Default: `plain`

Werte: `plain`, `bold`, `italic`, `bold-italic` oder eigener Bezeichner für einen `<font>`.

FontColor (Color) Schriftfarbe

Default: `black` (vgl. *Datentypen*)

> **Warning:** Transparenzen werden derzeit in ActionScript wie folgt unterstützt: Sofern ein Textwidget mit einem oder mehreren Textformaten formatiert ist, von denen eins oder mehrere eine Transparenz in der Farbangabe haben, wird das gesamte Textfeld auf diese Transparenz gesetzt. Dabei gilt der Transparenzwert des letzten verwendeten Formats mit einer Transparenz.
> Grund: In Actionscript kann kein Alpha-Wert auf Basis eines Textformats definiert werden, sondern dieser Wert kann nur für das gesamte Textfeld gesetzt werden.

Sekundäre Properties zur Schriftformatierung

VerticalOffset (Number) Vertikaler Offset für das Textelement. Arbeitet wie BaselineOffset, kann aber auch auf einzelne Worte angewandt werden.

Default: `0`

Underline (Boolean) (NYI) Text wird unterstrichen.

Default: `false`

UnderlineColor (Color) (NYI) Unterstreichungsfarbe.

Default: `black` (vgl. *Datentypen*)

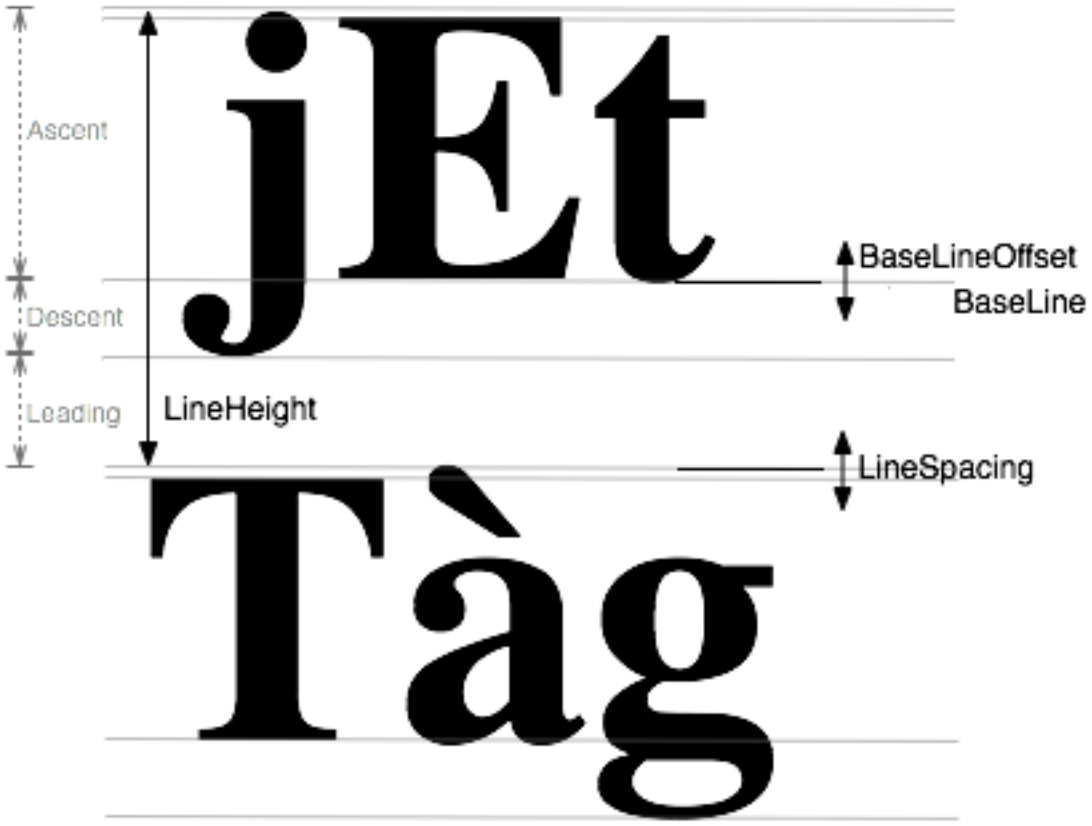
Hinweis: Diese Property wird in Actionscript nicht unterstützt (FlashPlayer 9)

UnderlineStyle (Symbol) (NYI) Stil (linie, doppelt, wellen, ..)

Hinweis: Diese Property wird in Actionscript nicht unterstützt (FlashPlayer 9)

Link (URL) (NYI) (NYI in Actionscript) Textabschnitt ist ein Link... Verhalten noch undefiniert.

Properties zur Absatzformatierung



Align (Symbol) Text-Alignment (left, right, center, justified)

Default: `left`

AutoBaseLine (Boolean, ReadOnly) (nur intern verwendet)

Default: `true`

BaseLine (Number) Mit `BaseLine` lässt sich die Position der Baseline auf einen absoluten Wert setzen. Angabe in Pixel.

Default: (automatisch). Setzen der `BaseLine` bewirkt, dass `AutoBaseLine` auf `false` gesetzt wird.

Hinweis: Diese Property wird in Actionscript nicht unterstützt (FlashPlayer 9)

BaseLineOffset (Number) Mit *BaseLineOffset* lässt sich die Baseline um einen relativen Wert nach oben/unten verschieben. Negative Werte schieben nach oben. Angabe in Pixel.

Default: `0`. Setzen des `BaseLineOffsets` bewirkt, dass `AutoBaseLine` auf `false` gesetzt wird.

Hinweis: Diese Property wird in Actionscript nicht unterstützt (FlashPlayer 9)

AutoLineHeight (Boolean, ReadOnly) (nur intern verwendet)

Default: `true`

LineHeight (Number) Die absolute Höhe einer Textzeile (in Pixel).

Modifikationen der BaseLine ändern die LineHeight nicht.

Default: (automatisch). Setzen der `LineHeight` bewirkt, dass `AutoLineHeight` auf `false` gesetzt wird.

Hinweis: Diese Property wird in Actionscript nicht unterstützt (FlashPlayer 9)

LineSpacing (Number) LineSpacing vergrössert/verkleinert die LineHeight. Negative Werte verringern die LineHeight. Angabe in Pixel.

Default: `0`

ParagraphSpacing (Number) Abstands nach einem Absatz. Angabe in Pixel.

Default: `0`

FirstLineIndent (Number) (NYI) Einrückung der ersten Zeile eines Absatzes. Angabe in Pixel.

Default: `0`

LineWidth (Number) (NYI) (NYI in Actionscript) Zeilenbreite. Angabe in Pixel.

Default: `null` (nicht gesetzt)

6.6.5 TextFormat-Maps

Mit <map> lassen sichj TextFormat-Objekte bündeln. Maps lassen sich auch vererben (`s2 extends s1`) (**NYI**):

```
<map name="s1">
        <textformat name="foo">
                <property ... />
        </textformat>
        <textformat name="bar">
                <property ... />
        </textformat>
</map>

<map name="s2" extends="s1">
        <textformat name="boo">
                <property ... />
        </textformat>
        <textformat name="bar">
                <property ... />
        </textformat>
</map>
```

Ein Styles-Objekt kann der TextStyles-Property zugewiesen werden, um dann die darin definierten TextFormate im Markup zu verwenden (`Formatter == 'markup'`).

6.6.6 Text-Markup

Note: Es ist geplant, einfache Textauszeichnungen zu unterstützen. Die Syntax für diese Auszeichnungen ist noch in der Entwicklung. Der Stand der Überlegungen (Jan 2010) ist folgender.

Kurzschreibweisen wie __text__ für bold können evtl. später eigeführt werden.

Es gibt folgende Sonderzeichen für Markup im Text: `|` und `[]`:

```
    |          Zeilenumbruch
    [ ... ]    Befehle, Styles, Sonstiges:

    [§]                    Neuer Absatz
    [§ myfmt]              Neuer Absatz mit Absatzformat myfmt (**NYI**)
    [mystyle hello world]  hello world wird mit Style 'mystyle' ausgezeichnet
    [*]                    Bullet (Neue Zeile und Listenpunkt) (**NYI**)
    [>]                    Tab (**NYI**)
    [p mein plain text]    Plain (**NYI**)
    [b mein fetter text]   Bold
    [i mein italic text]   Italic
    [s bold+italic text]   BoldItalic (**NYI**)
    [n 1234556]            Numberfont (**NYI**)
[d kjsfhdkhf]          Dingbats (**NYI**)
```

Um die Sonderzeichen selbst im Textfeld verwenden zu können, müssen sie mit \ escaped werden:

```
\| ist |
\[ ist [
\] ist ]
\_ ist _
```

] darf innerhalb einer Markup-Sequenz nicht verwendet werden.

Beispiel für Text-Markup:

```
<styles name="myStyles">
        <textformat name="foo">
                <property name="Font" value="MyFont1" />
        </textformat>
        <textformat name="bar">
                <property name="Font" value="MyFont2" />
        </textformat>
</styles>

<widget name="myWidget" Shape="R 400,50">
        <property name="MultiLine" value="true" />
        <property name="TextFormat" lookup="myStyles.bar" />
        <property name="TextStyles" lookup="myStyles" />
        <property name="Text">[§ foo]Dieser Absatz wird in MyFont1 dargestellt.
        [bar Nur hier wird MyFont2 verwendet.] Und hier geht es wieder in MyFont1
        weiter.[§ foo]Das ist schon der zweite Absatz, dargestellt in MyFont1.[§]
        Diese Zeile hat kein Absatzformat zugewiesen und wird daher durch die
        Angabe in der Property TextFormat in MyFont2 dargestellt.</property>
</widget>
```

6.6.7 Formatter

Der ausgewählte Formatter bestimmt, was das TextWidget darstellt.

plaintext (NYI) Der Text der `Text`-Property wird 1:1 dargestellt.

markup Der Text der `Text`-Property wird dargestellt. Text-Markup wird interpretiert.

printf (NYI) Die `Text`-Property dient als Formatierungsstring für eine Textformatierung entsprechend dem `printf`-Kommando. Als Value kann ein einzelner Wert oder eine Liste von Werten gesetzt werden, die als Input für die Formatierung dienen.

Beispiel:

```
<widget type="Text">
        <property name="Formatter" value="printf" />
        <property name="Text" value="Mehrere Werte darstellen:
        Ein String: %s, noch ein String: %s,\nund zwei Zahlenwerte:
        %2.1f (1 Nachkommastelle),\n%05d (mit Nullen gepaddet)." />
        <property name="Value">
                <list>
                        <item>Hans</item>
                        <item>Dieter</item>
                        <item type="Number">36.768</item>
                        <item type="Integer">13</item>
                </list>
        </property>
</widget>
```

Dieses Widget soll den folgenden Text anzeigen:

```
Mehrere Werte darstellen: Ein String: Hans, noch ein String: Dieter,
und zwei Zahlenwerte: 36.8 (1 Nachkommastelle),
00013 (mit Nullen gepaddet).
```

Zahlen- und Datums-Formatter Die Formatter setzen den `Text` des Widgets, indem der Value gemäß der Formatierungsanweisung formatiert wird.

Zur Erläuterung der Formatierungsanweisungen vgl. java.text.DecimalFormat (http://java.sun.com/javase/6/docs/api/java/text/DecimalFormat.html) bzw. java.text.SimpleDateFormat (http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html).

Liste der möglichen Formatter-Werte:

Formatter	Formatierungsanweisung (deutsche Dezimal-/Tausenderkennzeichnung)
integer	"#.##0"
integer2	"00"
integer3	"000"
decimal	"#.##0,0"
decimal0	"#.##0,#"
decimal1	"#.##0,0"
decimal2	"#.##0,00"
decimal3	"#.##0,000"
difference0	"+#.##0;-#.##0"
difference	"+#.##0,0;-#.##0,0"
difference1	"+#.##0,0;-#.##0,0"
difference2	"+#.##0,00;-#.##0,00"
difference3	"+#.##0,000;-#.##0,000"
percent	"#.##0,0%"
percent0	"#.##0%"
percent1	"#.##0,0%"
percent2	"#.##0,00%"
percent3	"#.##0,000%"
percentdiff0	"+#.##0%;-#.##0%"
percentdiff	"+#.##0,0%;-#.##0,0%"
percentdiff1	"+#.##0,0%;-#.##0,0%"
percentdiff2	"+#.##0,00%;-#.##0,00%"
percentdiff3	"+#.##0,000%;-#.##0,000%"

```
percentsign          "#.##0,0'%'"
percentsign0         "#.##0'%'"
percentsign1         "#.##0,0'%'"
percentsign2         "#.##0,00'%'"
percentsign3         "#.##0,000'%'"
psdiff0              "+#.##0'%';-#.##0'%'"
psdiff               "+#.##0,0'%';-#.##0,0'%'"
psdiff1              "+#.##0,0'%';-#.##0,0'%'"
psdiff2              "+#.##0,00'%';-#.##0,00'%'"
psdiff3              "+#.##0,000'%';-#.##0,000'%'"

euro                 "€ #.##0;€ -#.##0"
dollar               "$ #.##0;$ -#.##0"
euro2                "€ #.##0,00;€ -#.##0,00"
dollar2              "$ #.##0,00;$ -#.##0,00"
pound                "£ #.##0;£ -#.##0"
lira                 "YTL #.##0;YTL -#.##0"
yen                  "¥ #.##0;¥ -#.##0"
pound2               "£ #.##0,00;£ -#.##0,00"
lira2                "YTL #.##0,00;YTL -#.##0,00"
yen2                 "¥ #.##0,00;¥ -#.##0,00"
thousands            "#.##0,## 'Tsd.'"
millions             "#.##0,## 'Mio.'"
billions             "#.##0,## 'Mrd.'"
trillions            "#.##0,## 'Bio.'"

hide                 ""

date-german          "dd.MM.yyyy"
date-german-short    "d.M.yy"
year                 "yyyy"
month-year           "M/yyyy"
month-yearshort      "M/yy"
monthlong-year       "MMMM yyyy"
monthshort-year      "MMM yyyy"
month-long           "MMMM"
month-short          "MMM"
weekday              "EEEE"
```

6.6.8 Layout

Note: Das nachfolgend beschriebene Layoutverhalten ist noch nicht implementiert

.

vgl. auch das Kapitel *Layout* im Abschnitt Widget.

Widgets in einem Layout müssen im Verlauf der Layout-Berechnung `PreferredBounds` bestimmen. Diese Berechnung ist beim TextWidget spezifisch je nach Darstellungsmodus. Die PreferredBounds können dabei vom Layout-Container nicht in jedem Fall hundertprozentig umgesetzt werden; sie sind im Layout-Prozess aber ein wichtiger Anhaltspunkt, wie das Widget gezeichnet werden soll.

Verhalten von einzeiligem Text in Layoutcontainern

Bei einzeiligen Text gibt das TextWidget als PreferredBounds ein Rechteck zurück, das so breit ist wie der voll ausgeschriebene Text plus horizontales Padding und so hoch wie die LineHeight plus vertikales Padding. Ist `FixedWidth` gesetzt, so wird die Breite stattdessen aus FixedWidth plus horizontalem Padding bestimmt.

Verhalten von mehrzeiligem Text in Layoutcontainern

Bei mehrzeiligem Text gibt es folgende Layout-Optionen:

- Als PreferredBounds wird die Breite des Widgets plus die zur Darstellung des Textes nötige Höhe zurückgeliefert, wenn weder `FixedWidth` noch `FixedLines/MaxLines` gesetzt sind. — - oder: Breite und Höhe des Textblocks werden genutzt. Ist in den Textformaten keine Breite angegeben, bricht der Text dabei nur an den eingefügten Zeilenumbrüchen und Absätzen um. — ???
- Ist `FixedWidth` gesetzt, so nutzt das TextWidget diesen Wert für die Breite und berechnet die Texthöhe entsprechend. Dieses Rechteck (zuzüglich Padding) wird als PreferredBounds zurückgegeben. Ist zusätzlich `FixedLines` oder `MaxLines` gesetzt, so wird als Höhe nur die zur Darstellung nötige Höhe zurückgeliefert.
- Ist `FixedLines` oder `MaxLines` gesetzt, aber nicht `FixedWidth`, so nutzt das TextWidget die Angabe, um den Text auf genau bzw. maximal die angegebene Zeilenzahl umzubrechen. Die Breite wird automatisch bestimmt.

6.6.9 HOWGO Linebreaking

- alle textelemente werden zu einer zeile zusammengebaut
- die maxima von ascent, descent und leading werden gespeichert.
- autolineheight ist maxascent+maxdescent+maxleading
- autobaseline ist bei maxascent

6.7 StackWidget

`maongo.core.toolkit.StackWidget extends Widget`

Das StackWidget dient dazu, aus einer Sammlung von CardWidgets genau eines zu einem gegebenen Zeitpunkt darzustellen.

Beispiel für einen einfachen Stack:

```
<presentation>
        <widget type="Stack">
                <property name="Index" value="0" />
                <widget type="Card">
                        <property name="Texture" value="bild1.png" />
                </widget>
                <widget type="Card">
                        <property name="Texture" value="bild2.png" />
                </widget>
        </widget>
        <widget type="Button">
                <action trigger="button-clicked">
                        <set property="Index" value="1" /> <!-- set-Syntax??? -->
                </action>
        </widget>
</presentation>
```

Dieser Stack enthält zwei statische Karten, die unterschiedliche Bilder anzeigen werden. Das Setzen der `Index`-Property des Stacks auf `0` bewirkt, dass bei Start der Presentation das erste definierte CardWidget angezeigt wird. Klick auf den Button bewirkt, dass die zweite Karte angezeigt wird.

Als Bestandteile des StackWidgets kommen nur CardWidgets in Frage. CardWidgets sind entweder direkt aktivierbar (`StaticCard == true`, Default-Modus), oder sie werden erst durch einlaufende Daten gefüllt und anzeigbar (`StaticCard == false`).

Das StackWidget bietet die Möglichkeit, Karten zu verwalten (`DataManager == true`, Default-Modus), diese gezielt per Index zu aktivieren und durch diese zu blättern. Daneben ist es auch möglich, den `DataManager` zu deaktivieren und das Routing direkt zum Anzeigen der Karten zu benutzen (jedes eintreffende Data wird direkt angezeigt).

Ist `DataManager == true`, so erlaubt die `Index`-Property, direkt eine Karte anhand der Position in der Liste der Kartenobjekte auszuwählen. Das JavaScript-Interface bietet außerdem Methoden zum Blättern. Die Property `Size` hält die Gesamtzahl der Karten.

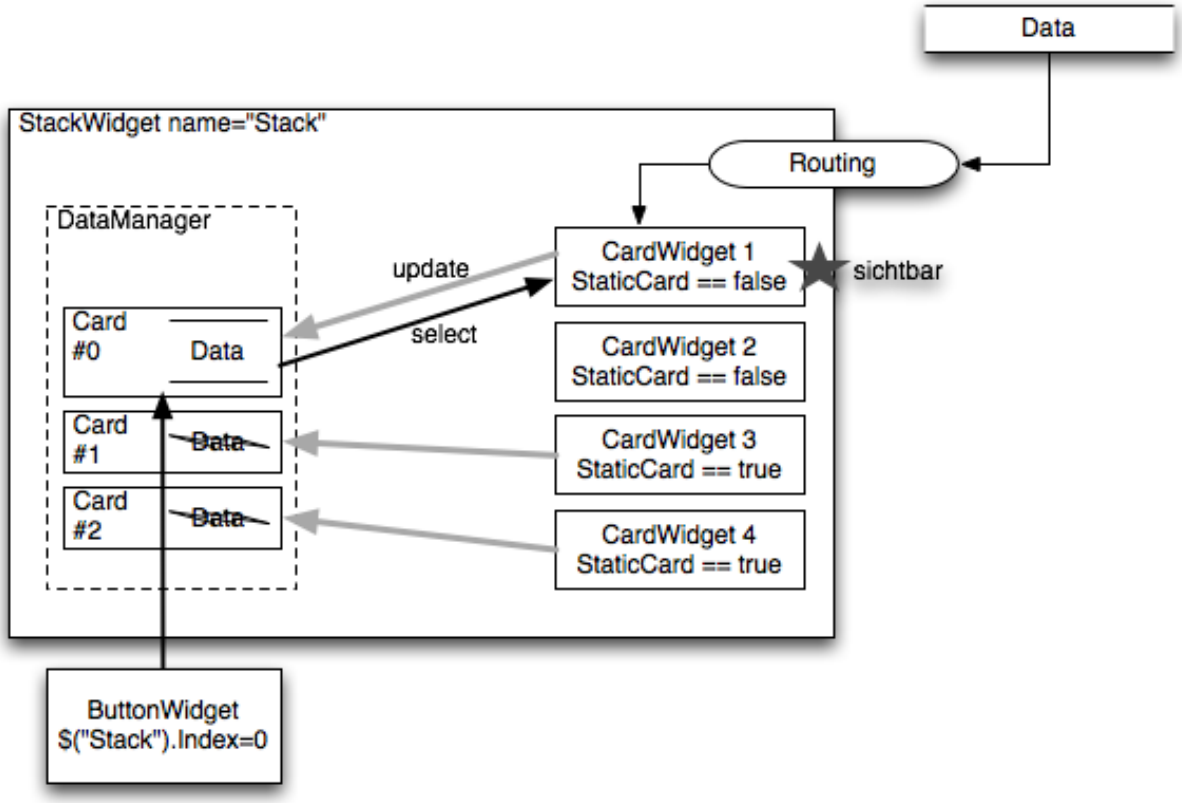


Abbildung für `DataManager == true`

Der Stack enthält zwei "statische Karten", die direkt vom DataManager verwaltet werden. Einlaufende Daten (Routing **muss** auf ein CardWidget erfolgen) werden ebenfalls vom DataManager verwaltet. In diesem Fall würde die Route auf CardWidget 1 zeigen.

Der eingezeichnete Button aktiviert Index 0, also die erste Karte im DataManager des Widgets. Das StackWidget schaltet evtl. sichtbare andere CardWidgets aus und das CardWidget 1 an.

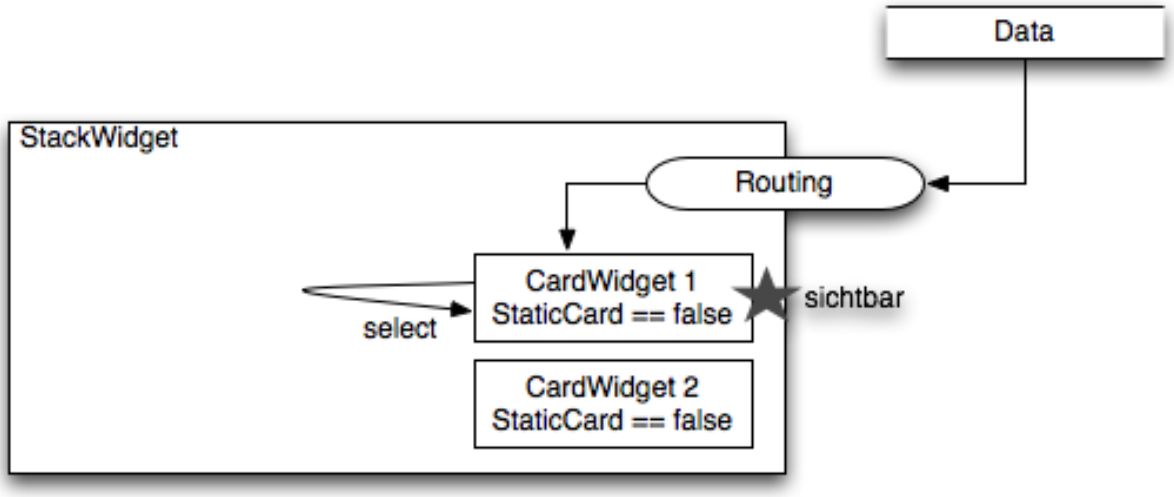


Abbildung für `DataManager == false`

Routing erfolgt auf Card-Widget 1. Es benachrichtigt den Stack (Ausschalten anderer CardWidgets) und wird automatisch vom Stack selektiert.

Beispiel mit Routing und Datamanager:

```
ToDo
```

Beispiel mit Routing, aber ohne Datamanager:

```
ToDo
```

6.7.1 Properties

StackMode (Symbol)

Schaltet zwischen den "Betriebsmodus" des StackWidgets um. 'managed' oder 'routing'

Default: `managed` (Das Widget hat eine interne Datenverwaltung und kann auch statische CardWidgets anzeigen.)

Index (Integer) Position der aktuell sichtbaren Karte in der internen Kartenliste (r/w). Die erste Karte hat den Index `0`. Das Sichtbarschalten einer Karte erfolgt durch Zuweisen eines Integers.

Default: -1 (keine Karte ausgewählt)

Liefert im `StackMode == single` null, wenn keine Karte angezeigt wird, und `0`, wenn eine Karte sichtbar ist.

Wird im `StackMode == stack` Index nicht gesetzt, so ist keine Karte sichtbar. Wird Index auf null gesetzt, so werden alle Cards unsichtbar geschaltet.

Wird ein negativer Wert zugewiesen, so wird Index auf 0 gesetzt. Wird ein Wert >= getTotalCards() zugewiesen, so wird Index auf getTotalCards() -1 gesetzt.

Size (Integer, read-only) Liefert die Anzahl der Kartenobjekte. 0, wenn keine Kartenobjekte vorhanden sind oder wenn `DataManager == false`.

SelectedCardWidget (Widget) Liefert das aktuell sichtbare CardWidget. Read-only.

Note: Sortierung: zurückgestellt

Ich schätze, statische Karten brauchen eine String-Property Sort, und für Data kann man mit SortField angeben, welche Data-Property zum Sortieren genutzt werden soll.

SortType könnte dann noch erlauben, numerisch oder alphanumerisch zu sortieren.

SortField

SortType

6.7.2 Methoden

Die folgenden Methoden sind nur bei `DataManager == true` sinnvoll einsetzbar.

showFirst() Aktiviert das erste Card-Objekt der internen Liste.

Entspricht `Index = 0`.

showLast() (nur bei `DataManager == true`)

Enspricht dem Setzen des `Index` auf den Wert `getTotalCards() -1`.

showNext() Aktiviert das nächste Card-Objekt der internen Liste.

Entspricht dem Hochzählen des Index um eine Position bis `getTotalCards() -1`.

showPrevious() Aktiviert das vorangehende Card-Objekt der internen Liste.

Entspricht dem Herunterzählen des Index um eine Position bis `0`.

Beispiel:

```
ToDo
```

6.7.3 Signals

card-activated Beim Setzen des Index und bei allen JavaScript-show*()-Funktionen, wenn dadurch eine neue Karte angezeigt wird.

card-deactivated Beim Setzen des Index und bei allen JavaScript-show*()-Funktionen, wenn dadurch ein CardWdiget deaktiviert (ausgeblendet) wird.

6.8 CardWidget

```
maongo.core.toolkit.CardWidget extends Widget
```

Das CardWidget ist stets Bestandteil eines Stacks.

Es hat besondere Funktionen innerhalb des Stacks:

Der Weg eines Data in den Stack beginnt auf dem CardWidget, das StackWidget hat keine besonderen Routingfunktionen. Das CardWidget gibt das Data zurück an den Stack mit der Information, welches CardWidget zuständig ist. Diese Information wird zusammen mit den Data an das DataManagement übergeben (wenn auf dem Stack `DataManager == true` ist, oder der Stack nutzt sie direkt, um das CardWidget zu aktivieren (wenn `DataManager == false`).

6.8.1 Properties

StaticCard (Boolean)

Zeige diese Card unabhängig von einlaufenden Data an.

Default: `true`

Schalten Sie StaticCard eines CardWidgets auf `false`, wenn Sie das entsprechende Widget nur zur Darstellung von einlaufenden Data nutzen wollen.

6.8.2 Methoden

n/a

6.8.3 Signals

n/a

Section author: malte

6.9 LineWidget

`maongo.core.toolkit.LineWidget extends Widget`

Das LineWidget dient dazu anhand einer Sammlung von Koordinaten eine Linie zu zeichnen. Eingesetzt wird es beispielsweise in den LineCharts zur Darstellung der einzelnen Linien. Wahlweise können an den Koordinaten Markierungen mittels eines Templates gezeichnet werden. Auch kann eingestellt werden, nur die Markierungen ohne verbindende Linien zu zeichnen.

Note: Eventuell macht es Sinn zu einem späteren Zeitpunkt eine interne Schachtelung einzuführen um einen "Container" um das LineWidget herum zu haben. Dieser könnte dann sowohl clippen als auch ggf. skalieren. Ebenfalls ist angedacht dem LineWidget einen Koordinatenraum zu übergeben.

Ein einfaches Beispiel ohne Markierungen an den Datenpunkten:

```
<widget name="myline" type="Line" location="40,40">
    <property name="BackgroundColor" value="yellow" />
    <property name="LineWidth" value="3" />
    <property name="Points" value="10,10; 50,100; 70,20" />
    <property name="DrawBobbles" value="false" />
</widget>
```

Ein einfaches Beispiel mit Default-Markierungen an den Datenpunkten:

```
<widget name="myline" type="Line" location="40,40">
    <property name="BackgroundColor" value="yellow" />
    <property name="LineWidth" value="3" />
    <property name="Points" value="10,10; 50,100; 70,20" />
    <property name="DrawBobbles" value="true" />
</widget>
```

Ein einfaches Beispiel mit selbst definierten Markierungen an den Datenpunkten:

```
<widget name="myline" type="Line" location="40,40">
    <property name="BackgroundColor" value="yellow" />
    <property name="LineWidth" value="3" />
    <property name="Points" value="10,10; 50,100; 70,20" />

    <template name="BobbleTemplate">
       <property name="BackgroundColor" value="red" />
       <property name="Shape" value="E -5 -5 10 10"/>
    </template>

</widget>
```

6.9.1 Funktionsweise

Das LineWidget zeichnet anhand der übergebenen Koordinaten und der sonstigen Linien-Properties ein Shape in sich selbst. Ein zusätzlich auf dem LineWidget gesetztes Shape wird ignoriert bzw. überschrieben.

Die Linie wird mit der `BackgroundColor` des Widgets gezeichnet. Ist eine Texture oder eine Outline (LineWidth etc.) angegeben, so werden sie für die Darstellung genutzt.

Da das Shape des LineWidgets durch die übergebenen Punkte/Properties definiert wird, haben diese auch Auswirkungen auf die Verhaltensweise des LineWidgets in einem Layout. So ändert sich bei einer Änderung der Punkte/Line-Properties auch das

Sofern innerhalb des LineWidgets ein Template mit dem Namen "BobbleTemplate" vorhanden ist, wird dieses genutzt, um die Datenpunkte zu markieren.

Hierfür wird an jedem Datenpunkt eine Instanz des Templates platziert. Dieses Verhalten kann über die Property `DrawBobbles` gesteuert werden.

Sofern die Property `DrawBobble` auf `true` gesetzt ist und keine Template existiert, werden Default-Bobbles (Kreis mit einem Durchmesser von 10 Pixel) in der Linienfarbe gezeichnet.

Das LineWidget wird auch innerhalb des LineChartWidgets genutzt, siehe *LineChartWidget @*.

6.9.2 Properties

Eigene Properties:

Points (List(Point): Liste mit Punkten) Liste der Koordinatenpunkte, die als Linie dargestellt werden sollen.

Default: `leere Liste`

Wie wird der Koordinatenraum festgelegt? Vermutlich transformiert das Widget die Punktangaben, zumindest in Y-Richtung (sonst müssten wir für positive Werte negative Punktangaben eingeben)

DrawBobble (Boolean) Diese Property legt fest, ob Datenpunkte mit dem in "BobbleTemplate" festgelegten Widget markiert werden. Falls diese Property auf "false" gesetzt wird, werden die Datenpunkte nicht markiert. Wird diese Property auf `true` gesetzt, wird das Clipping des LineWidgets deaktiviert um die Bobbles, welche als Child-Widgets des LineWidgets platziert werden, sichtbar zu machen.

Default: `true`

LineWidth (Number) Stärke der Linie die gezeichnet wird

Default: `1.0`

LineCap (Symbol) ***NYI*** Art wie Ecken (inkl. Start- und Endpunkt) der Linie gezeichnet werden.

Default: `symbol...`

DrawLine (Boolean) Diese Property legt fest, ob zwischen den Punkten eine Linie gezeichnet werden soll oder nicht.

Default: `true`

Note: weitere angedachte Properties: Dashes, FillStyles.

Section author: jo

Note: XXXXXXXXXXXXXXX Status 18.10.2010: vorläufig. XXXXXXXXXXXXXXXXXXXX

6.10 CanvasWidget

`maongo.core.toolkit.CanvasWidget extends Widget`

6.10.1 Funktionsweise

6.10.2 Properties

Eigene Properties:

Effect (Symbol)

Section author: jo

CHAPTER

SEVEN

MAONGO-CHARTS

Section author: jo, benjamin

7.1 Getting Started

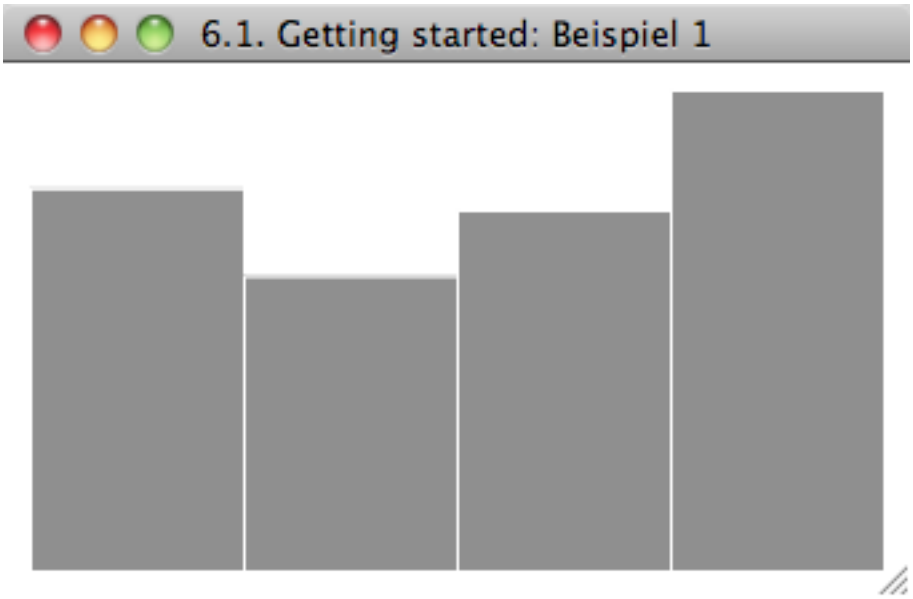
MaongoMP bietet für das automatische Platzieren von Labels und für die Anbindung von dynamischen Daten und daraus resultierenden automatischen Layouts ein ganzes Subsystem an Optionen an, die es erlauben, so gut wie jeden Gestaltungswunsch umzusetzen.

Um allerdings einfach loslegen zu können, lohnt es sich, zunächst einmal ein Minimalchart ohne diese Features zu bauen.

7.1.1 Minimalistisch...

Ein einfaches BalkenChart könnte so erzeugt werden:

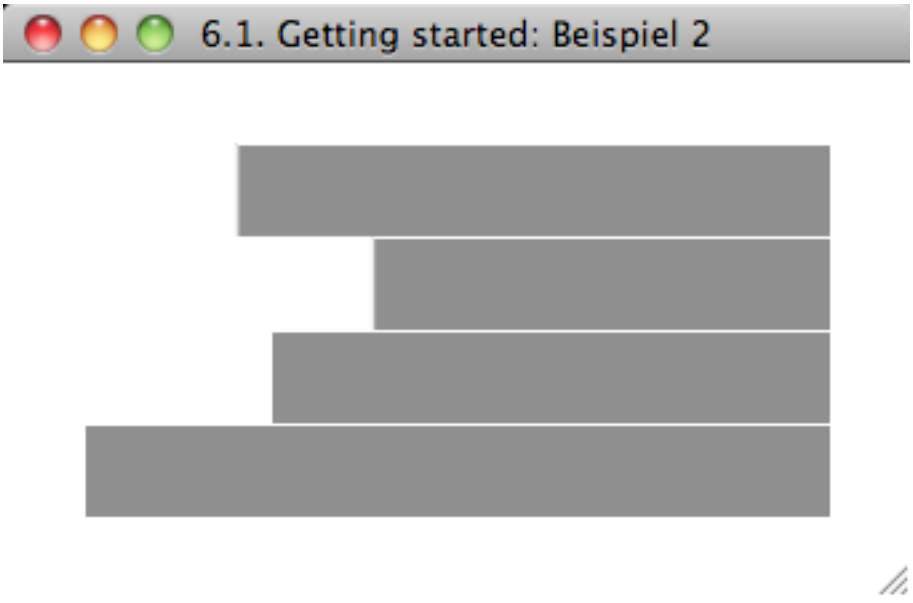
```
Beispiel 1:

<widget name="HelloWorldBars" type="BarChart" x="10" y="10" shape="R 320 180">
  <property name="Values" type="List[Number]" value="3.5,2.7,3.3,4.4" />
  <property name="ForegroundColor" value="gray" />
</widget>
```



Das Barchart nutzt die Liste von vier zugewiesenen Werten, um Balken in der (angegebenen oder ererbten) `ForegroundColor` zu zeichnen. Es ordnet die Balken automatisch innerhalb des durch `Shape` und `Padding` angegebenen Bereichs an. Dasselbe funktioniert auch horizontal:

`Beispiel 2:`

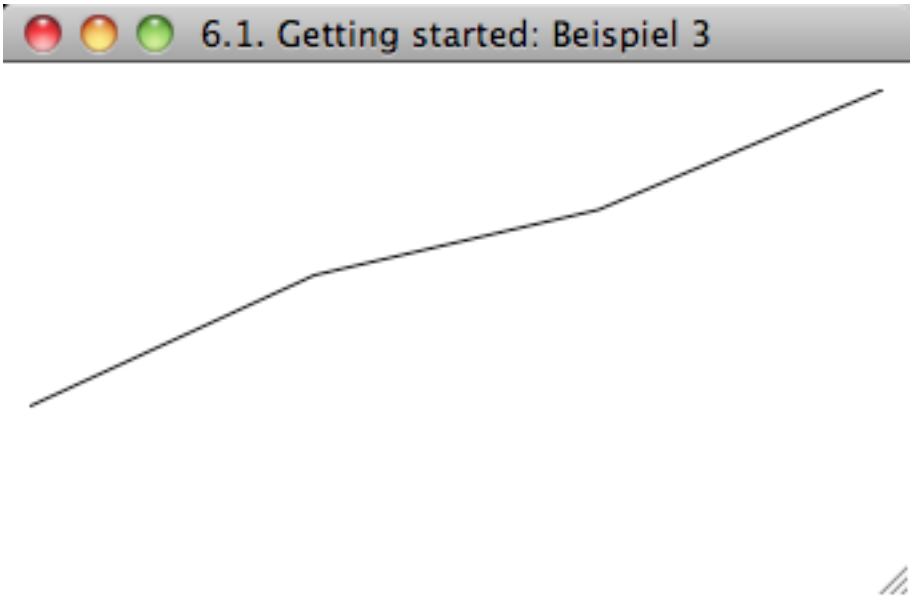
```
<widget name="HelloWorldBars" type="BarChart" x="10" y="10" shape="R 320 180">
  <property name="Padding" value="20" />
  <property name="Values" type="List[Number]" value="3.5,2.7,3.3,4.4" />
  <property name="ForegroundColor" value="gray" />
  <property name="Orientation" value="left" />
</widget>
```



Eine minimalistische Umsetzung eines Linecharts wäre:

Beispiel 3:

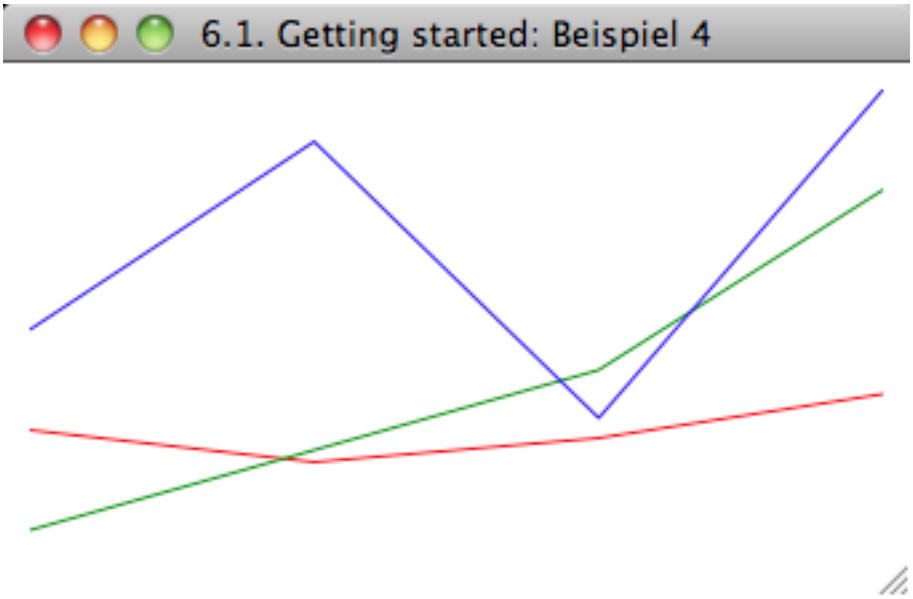
```
<widget name="HelloWorldLine" type="LineChart" shape="R 320 180">
  <property name="Values" type="List[Number]" value="[[3.5,2.7,3.3,4.4]]" />
  <property name="ForegroundColor" value="gray" />
</widget>
```



Das Beispiel skaliert die Werte im verfügbaren Raum und zeichnet eine Linie in der `ForegroundColor` mit den in `Values` angegebenen Werten. Da eine Linie aus einer Liste von Werten besteht, und in einem LineChart mehrer Linien gezeichnet werden können, werden die `Values` als Liste von Listen (für jede Linie eine Liste) notiert. Hier ein LineChart mit drei Linien:

Beispiel 4:

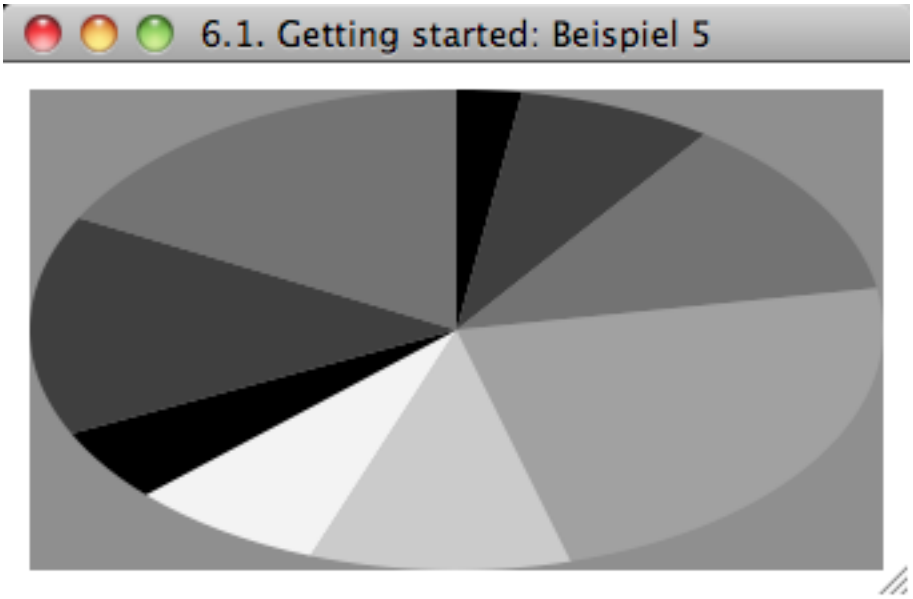
```
<widget name="HelloWorldLine" type="LineChart" shape="R 320 180">
  <property name="Values" type="List[Number]" value="[[3.5,2.7,3.3,4.4],[1,3,5,9.5],[6,1.7,3.8,12]]"
  <property name="Colors" type="List[Color]" value="[red, green, blue]" />
</widget>
```



Die Farben der Linien werden mit einer Liste von Farben in der Property `Colors` gesetzt. Wenn mehr Linien als Farbeinträge vorhanden sind, wird wieder von vorne aus der Liste ausgelesen.

Beispiel 5 zeichnet eine minimalistische Tortengrafik:

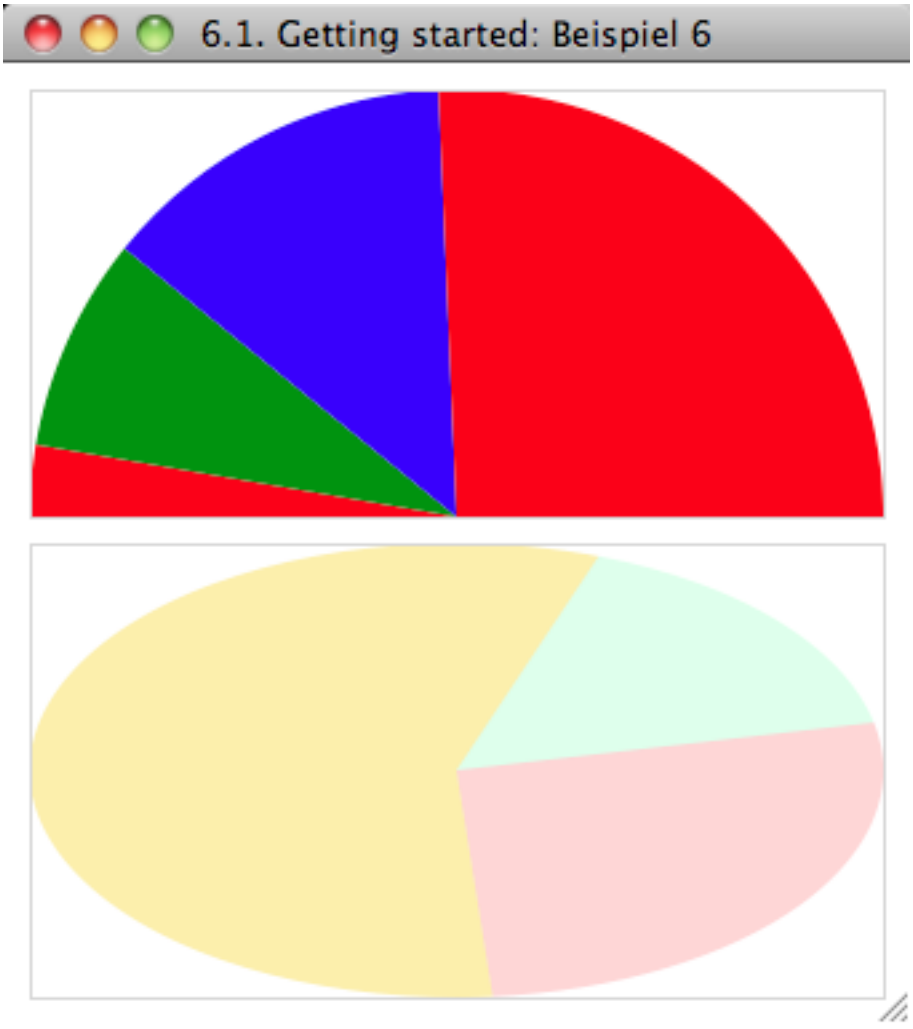
```
Beispiel 5:

<widget name="HelloWorldPie" type="PieChart" shape="R 320 180">
  <property name="Values" type="List[Number]" value="1,3,5,9.5,4,3,2,6,7" />
      <property name="ForegroundColor" value="gray" />
  <property name="BackgroundColor" value="gray" />
</widget>
```



Auch hier werden die Standardeinstellungen des Charttyps genutzt, um ein Kreischart zentriert in die Mitte des verfügbaren Raums zu zeichnen.

Mit geringen Modifikationen können wir auch ein Halbkreis- und eine ellipsoides Chart zeichnen:

```
Beispiel 6:

<widget name="HelloWorldPie2" type="PieChart" x="10" y="10" shape="R 320 180">
  <property name="BorderWidth" value="1" />
  <property name="Values" type="List[Number]" value="1,3,5,9.5" />
  <property name="Colors" type="List[Color]" value="[red, green, blue]" />
  <property name="PieMode" value="seats" />
</widget>

<widget name="HelloWorldPie3" type="PieChart" x="10" y="200" shape="R 320 180">
  <property name="BorderWidth" value="1" />
  <property name="Values" type="List[Number]" value="1,3,5,9.5" />
  <property name="Colors" type="List[Color]" value="[#ffcc00/66, #33ff88/33, #ff0000/33]" />
  <property name="PieMode" value="ellipse" />
</widget>
```



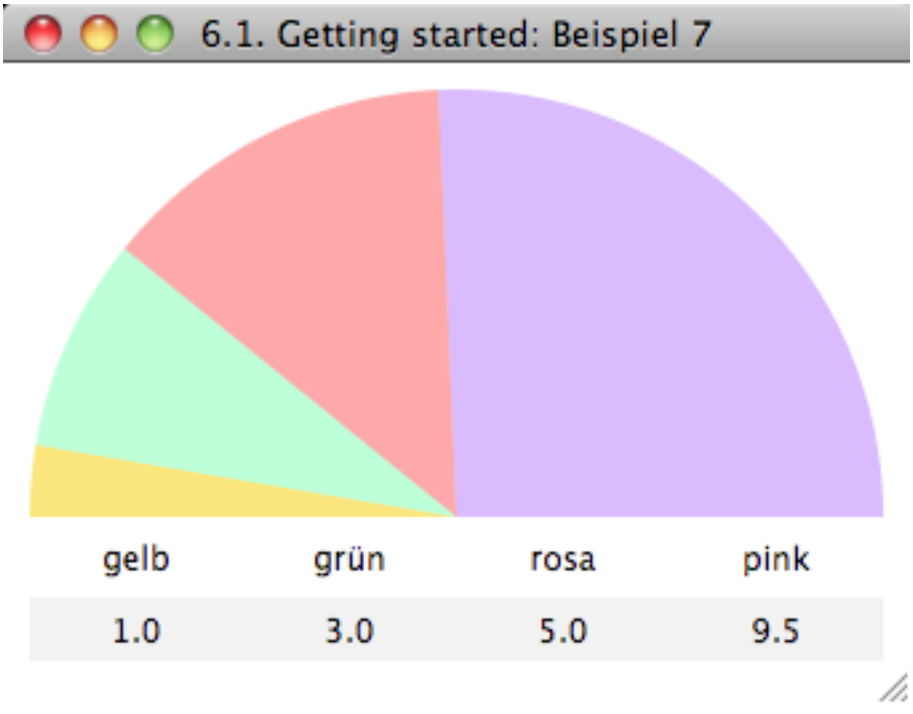
Auch in diesem Beispiel haben wir die Property `Colors` genutzt, um die einzelnen Segmente einzufärben. Im ellipsoiden Chart haben wir den Farben zusätzlich Transparenz gegeben.

Die bisher beschriebenen minimalistischen Ansätze reichen häufig nicht aus, um komplexere Charts zu erstellen: wir wollen Balken, Linien und Kreissegmente beschriften, wollen Data (siehe –> Data) nutzen, um Werte des Charts und der Labels zu setzen, und wollen häufig auch die Labels in spezifischer Art und Weise um die Charts gruppieren oder an Werte des Charts koppeln.

Bevor wir solch komplexe Charts erstellen, hier noch eine ziemlich minimalistische Version eines PieCharts mit einfachen Labels:

Beispiel 7:

```
<!-- Das PieChart -->
<widget name="ThePie" type="PieChart" x="10" y="10" shape="R 320 180">
  <property name="Values" type="List[Number]" value="1,3,5,9.5" />
  <property name="Colors" type="List[Color]" value="[#ffcc00/99, #33ff88/66, #ff0000/66, #8833ff/66]"
  <property name="PieMode" value="seats" />
</widget>

<!-- Die erste LabelBar -->
<widget name="FirstLabels" type="LabelBar" x="10" y="190" shape="R 320 20">
  <template name="LabelTemplate" type="Text" shape="R 80 35"/>
  <property name="Values" type="List[String]" value="Wert1,Wert2,Wert3,Wert4" />
  <property name="LabelTemplates" value="[@LabelTemplate]"/>
</widget>

<!-- Ein Template außerhalb der Labelbar -->
<template name="EinAnderesLabelTemplate" type="Text" shape="R 80 35">
  <property name="BackgroundColor" value="#dddddd" />
</template>

<!-- Die zweite LabelBar -->
<widget name="SecondLabels" type="LabelBar" x="10" y="210" shape="R 320 20">
  <property name="Values" type="List[Number]" value="1,3,5,9.5" />
  <property name="LabelTemplates" value="[@EinAnderesLabelTemplate]" />
</widget>
```



Hier haben wir unter dem PieChartWidget zwei LabelBarWidgets angeordnet, die für jedes ihrer Values ein neues Element aus einem Template generieren. Die Templates, die in einem LabelBar benutzt werden sollen, müssen als Liste aus Template-Referenzen in der Property `LabelTemplates` gesetzt werden. Die zweite LabelBar zeigt, daß das benutzte Template auch an anderer Stelle im Mad stehen kann. Mehr zu Templates und ihrer Benutzung im

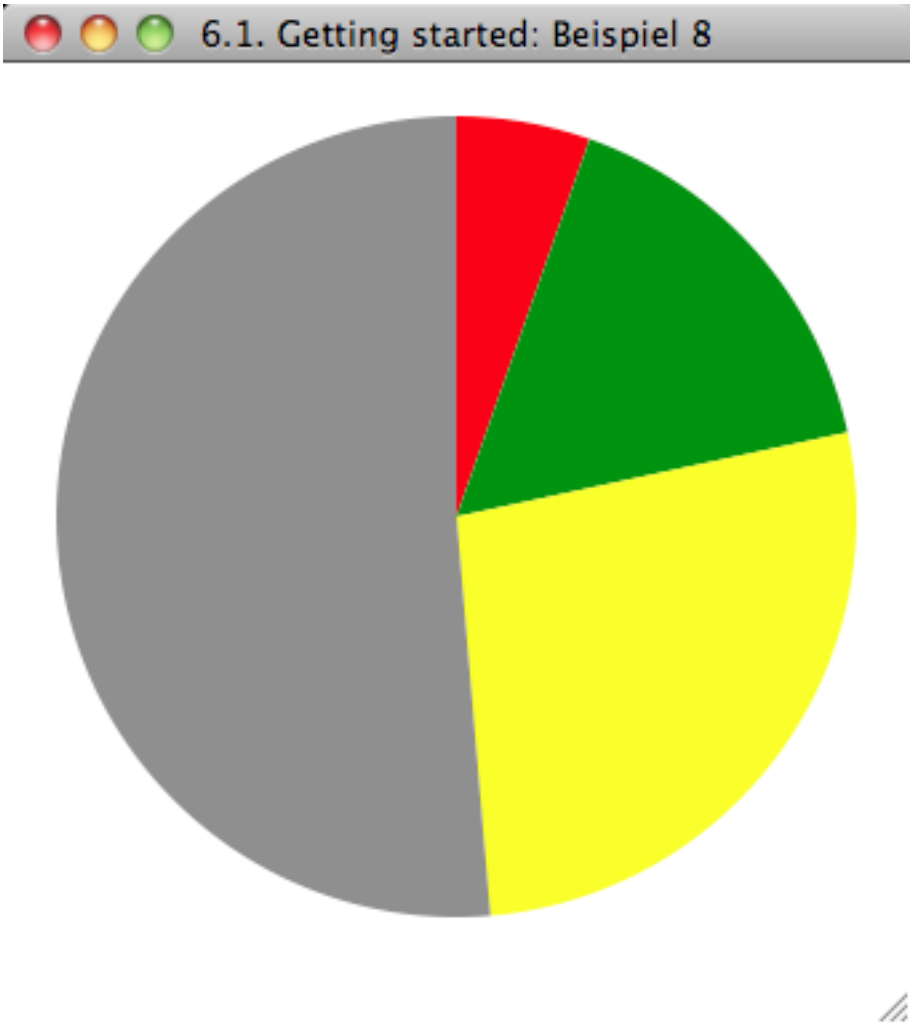
folgenden Absatz:

7.1.2 Templates nutzen

Alle Charts können für die wiederholten Elemente (Balken, Linien, Kreissegmente, Labels) Templates nutzen. Templates sind ja bekanntlich (–> *Templates*) beliebige Widgets, die bei der Ausführung der Presentation initialisiert (und im Fall eines Charts mehrfach automatisch eingesetzt) werden.

Wenn wir das Beispiel 5 von oben so modifizieren wollen, dass jedes Kreissegment eine anderes Template mit unterschiedlichen Farben enthält, so können wir so vorgehen:

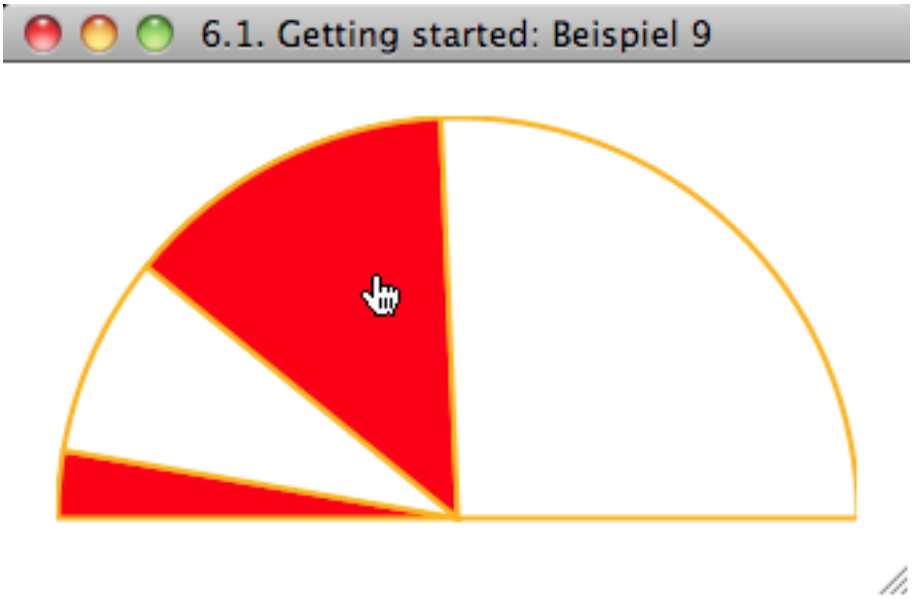
```
Beispiel 8:

      <template name="red" type="Widget">
        <property name="BackgroundColor" value="red" />
      </template>

      <template name="green" type="Widget">
        <property name="BackgroundColor" value="green" />
      </template>

      <template name="yellow" type="Widget">
        <property name="BackgroundColor" value="yellow" />
      </template>

      <template name="gray" type="Widget">
        <property name="BackgroundColor" value="gray" />
      </template>

      <widget name="ColorPie" type="PieChart" x="20" y="20" shape="R 300 300">
        <property name="Values" type="List[Number]" value="1,3,5,9.5" />
        <property name="ForegroundColor" value="gray" />
        <property name="PieTemplates" type="List[String]" value="[@red,@green,@yellow,@gray]" />
      </widget>
```



Die Property `PieTemplates` erhält hier eine Liste von Templatenamen zugewiesen, die für das PieChartWidget per Lookup erreichbar sein müssen. Es nutzt diese Templates der Reihe nach, um die Werte damit als Kreissegmente zu zeichnen.

Ebenso könnten wir aber auch entscheiden, dass ein Template mehrfach genutzt werden soll. Auch ein Button kann hier als Template verwendet werden:

```
Beispiel 9:

<template name="mybutton" type="Button">
  <property name="BorderWidth" value="2" />
  <property name="BorderColor" value="orange" />
  <action trigger="button-clicked">
    this.BackgroundColor = "red";
  </action>
</template>

<widget name="ClickablePie" type="PieChart" x="20" y="20" shape="R 300 300">
  <property name="Values" type="List[Number]" value="1,3,5,9.5" />
  <property name="PieTemplates" type="List[String]" value="[@mybutton]" />
  <property name="PieMode" value="seats" />
</widget>
```



Hier wird das Template mybutton für die Darstellung aller Segmente benutzt. Ein Klick auf ein Kreissegment färbt es rot.

7.1.3 Chart und Templates an Data binden

Bereits Beispiel 8 und 9 wecken weitere Wünsche: Es wäre sehr praktisch, unser Chart und die zum Zeichnen verwendeten Templates an die Werte eines auf das Widget gerouteten Data binden zu können.

Angenommen, wir haben ein Data, das auf einem Button definiert ist und so aussieht:

Beispiel 10:

```
<widget name="Databutton" type="Button" x="200" y="140" shape="O 100 50">
  <property name="Text" value="Route Data!" />
  <route datatype="*" target="BC_CTRL"/>
  <property name="DataMode" value="store-forward"/>
  <property name="Data">
    <data type="SimpleData" key="mydata">
      <table name="Table">
        <row>
          <cell>Hans</cell>
          <cell type="Color">#cc0000</cell>
          <cell type="Decimal">1.0</cell>
          <cell type="Decimal">-1.0</cell>
          <cell>Gruppe 1</cell>
          <cell type="Image">../images/hans.png</cell>
        </row>
        <row>
          <cell>Peter</cell>
          <cell type="Color">#00cc00</cell>
          <cell type="Decimal">3.0</cell>
          <cell type="Decimal">-1.5</cell>
          <cell>Gruppe 1</cell>
          <cell type="Image">../images/peter.png</cell>
        </row>
        <row>
```

```
          <cell>Dieter</cell>
          <cell type="Color">#0000cc</cell>
          <cell type="Decimal">5.0</cell>
          <cell type="Decimal">0.5</cell>
          <cell>Gruppe 2</cell>
          <cell type="Image">../images/dieter.png</cell>
        </row>
        <row>
          <cell>Georg</cell>
          <cell type="Color">#cc00cc</cell>
          <cell type="Decimal">9.5</cell>
          <cell type="Decimal">3.0</cell>
          <cell>Gruppe 2</cell>
          <cell type="Image">../images/georg.png</cell>
        </row>
      </table>
    </data>
  </property>
</widget>
```

Ein Klick auf den Button leitet das Data zum Widget `TheChart` weiter:

Beispiel 10 cont'd:

```
<engine type="ChartController" name="BC_CTRL">
  <property name="ColumnNames" value="names,colors,values,diffs,groups,images"/>
  <property name="DataTableView" value="Table" />
</engine>

<widget name="TheChart" type="BarChart" x="10" y="10" shape="R 320 120">
  <property name="Controller" value="@BC_CTRL"/>
  <bind property="Values" to="Controller.Columns" key="values" />
  <bind property="Groups" to="Controller.Columns" key="groups" />
  <property name="GroupMetrics" value="spacing: 10" />
      <property name="BarTemplates" type="List[String]" value="[@BarTempl]" />
  <template name="BarTempl" type="Widget">
    <bind property="Texture" to="ControllerProxy" key="images" />
    <property name="TextureMode" value="top-center" />
    <bind property="BackgroundColor" to="ControllerProxy" key="colors" />
  </template>
</widget>
```

Wir sehen, dass es Properties gibt, denen Wertelisten zugewiesen werden, da sie für das ganze Chart gelten: `Values` und `Groups` (letztere erlaubt uns, mit `GroupMetrics` die Balken des BarCharts zu gruppieren).

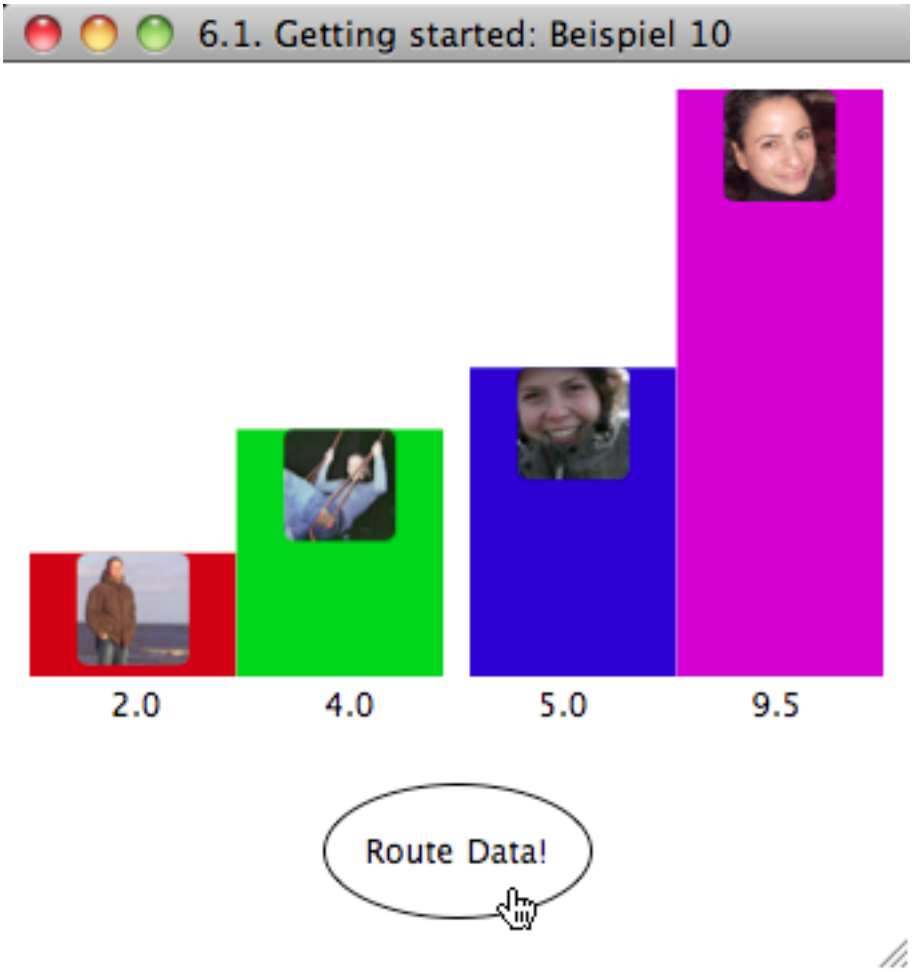
Daneben gibt es Properties im Template, die wir an einen bestimmten Wert einer Liste binden wollen. Dazu dient das Objekt "Controller.current": beim Erzeugen des Charts wird für jeden Wert eine Kopie des Templates erzeugt. Diese Kopie wird mit den Binding-Angaben (hier für `Texture` und `BorderColor` an den gerade aktuellen Index (= `current`) des Controller-Objekts gebunden und erhält den unter `key` verfügbaren Wert, hier also eine Farbe und ein Bild.

Die LabelBars können sich an den Controller des PieCharts binden und von dort Ihre `Values` füllen.

Nun wollen wir auch endlich die LabelBar für die Zahlenwerte mit einem Textwidget formatieren. In diesem Fall binden wir nicht nur die Werteliste, sondern im Template auch die einzelnen Werte mit `current`:

Beispiel 10 cont'd:

```
<widget name="TheLabels2" type="LabelBar" x="10" y="230" shape="R 320 20">
  <property name="Gap" value="2" />
  <property name="Controller" value="@BC_CTRL"/>
  <bind property="Values" to="Controller.Columns" key="values" />
  <property name="LabelTemplates" type="List[String]" value="[@LabelBarTempl]" />

  <template name="LabelBarTempl" type="Text">
    <property name="TextAlign" value="center" />
    <bind property="Text" to="ControllerProxy" key="values" />
  </template>
</widget>
```



7.1.4 Raster und Layout zur Platzierung der Labels nutzen

In manchen Situationen ist es nötig, die Positionierung der Labels nicht unabhängig vom (Balken-, Linien-)Chart vorzunehmen, sondern an die Positionen im gezeichneten Chart zu koppeln. Die LabelBars, die wir in den bisherigen Beispielen mit den Charts verwendet haben, bieten diese Möglichkeit nicht: sie sind unabhängige Widgets, die ihre eigene Layout-Funktionalität nutzen.

In einem gruppierten Säulenchart werden die Säulen auseinander gerückt: Die Labels sollten dem folgen. Dies erreichen wir, indem wir im BarChart ein einfaches Raster für die Labels definieren und mit LayoutConstraints die Labelpositionierung festlegen.

```
Beispiel 11:
Databutton wie oben

<engine type="ChartController" name="BC_CTRL">
  <property name="ColumnNames" value="names,colors,values,diffs,groups,images" />
  <property name="DataTableView" value="Table" />
</engine>

<widget name="TheChart" type="BarChart" x="10" y="10" shape="R 320 220">
  <!-- Die Referenz auf den ChartController -->
  <property name="Controller" value="@BC_CTRL" />

  <!-- Binden der Werte an den Controller -->
  <bind property="Values" to="Controller.Columns" key="values" />
  <bind property="Groups" to="Controller.Columns" key="groups" />

  <!-- Abstände zwischen Balken und Gruppen -->
  <property name="BarMetrics" value="spacing: 3"/>
  <property name="GroupMetrics" value="spacing: 20" />

  <!-- Definition des horizontalen und vertikalen Rasters -->
  <property name="HorizontalRaster" type="List[KeyValue]" value="[left: 9, chart: max, right: 9]" />
  <property name="VerticalRaster" type="List[KeyValue]" value="[top: 6, textlabel: 20, valuelabel: 2(

  <!-- Definition der zu nutzenden Templates -->
  <property name="BarTemplates" type="List[String]" value="[@BarTempl]" />
  <property name="BarLabels" type="List[String]" value="[@Wertlabel, @Textlabel]" />
</widget>
```

Wir haben hier ein BarChart soweit vorbereitet wie in Beispiel 10 das PieChart.

Neu hinzugekommen ist die Definition eines horizontalen und eines vertikalen Rasters. Was noch fehlt, ist die in ElementLabels genannten Templates zu definieren und in dieser Definition die Verortung im Raster zu benennen.

Die Balken werden per Konvention in der Rasterposition `chart-chart` (also `HorizontalRasterPosition=chart` und `VerticelRasterPosition=chart`) gezeichnet.

Mit der Property `GroupMetrics` haben wir außerdem festgelegt, dass zwischen den beiden Gruppen von Balken ein Abstand von 20px gezeichnet werden soll. `BarMetrics` sorgt für den Abstand zwischen den Balken.

Definition der Templates für die Labels und die Balken:

```
Beispiel 11 cont'd:

<template name="Textlabel" type="Text">
  <bind property="Text" to="ControllerProxy" key="names" />
  <property name="TextAlign" value="center" />
  <property name="LayoutConstraints">
    <map type="whatever">
      <property name="VerticalRasterPosition" value="textlabel" />
      <property name="HorizontalRasterPosition" value="chart" />
      <property name="XSource" value="anchor" />
      <property name="YSource" value="raster" />
      <property name="WidthSource" value="raster" />
      <property name="HeightSource" value="raster" />
      <property name="Align" value="top-center" />
    </map>
  </property>
```

```
</template>
<template name="Wertlabel" type="Text">
  <bind property="Text" to="ControllerProxy" key="values" />
  <property name="TextAlign" value="center" />
  <property name="BackgroundColor" value="#efefef" />
  <property name="LayoutConstraints">
    <map type="whatever">
      <property name="VerticalRasterPosition" value="valuelabel" />
      <property name="HorizontalRasterPosition" value="chart" />
      <property name="XSource" value="anchor" />
      <property name="YSource" value="raster" />
      <property name="WidthSource" value="raster" />
      <property name="HeightSource" value="raster" />
      <property name="Align" value="top-center" />
    </map>
  </property>
</template>
<template name="BarTempl">
  <bind property="BackgroundColor" to="ControllerProxy" key="colors" />
  <property name="LayoutConstraints">
    <map type="whatever">
      <property name="XSource" value="raster" />
      <property name="YSource" value="raster" />
      <property name="WidthSource" value="raster" />
      <property name="HeightSource" value="raster" />
    </map>
  </property>
</template>
```

Wir sehen, dass die Labels wie vorgegeben an unterschiedlichen vertikalen Positionen gezeichnet werden. Die Gruppierung wirkt sich in erster Linie auf die Säulen aus; da die Labels über das Raster aber an die Säulenpositionen gebunden sind, sehen wir bei ihnen denselben Gruppenabstand.

Nähere Informationen zum LayoutRaster und den möglichen LayoutConstraints siehe im Kapitel *Raster und Layout-Constraints*.

Test: See **this example script**.

Section author: jo, ben

Note: XXXXXXXXXXXXXXX Status 12.10.2010: verbindlich. XXXXXXXXXXXXXXXXXXXX

Note: XXXXXXXXXXXXXXX Status 20.10.2010: verbindlich, List[Lookup] o.ä. steht noch aus XXXXXXXXXXXXXXXXXXXX

7.2 Gemeinsame Properties aller Charttypen

Die im Paket MaongoCharts versammelten Widgettypen (*BarChartWidget*, *LineChartWidget @*, *PieChartWidget*, *LabelBarWidget*) haben einige gemeinsame Properties...

7.2.1 Properties

AnimationProgress (List[Number])

???

AnimationElements (List[Object]) (read-only) Die Einzelelemente des Charts (zur Verwendung in Animationen).

siehe: *Besonderheiten bei Chartanimationen*

Controller (ChartController) Der Controller steht den Elementen (Lines, Bars, Labels, ...) dieses Widgets über deren "ControllerProxy" Property zur Verfügung. Ist die "ValueColumn" property auf dem Controller gesetzt, dann bindet dieses Chart seine Value Property automatisch an den Controller.

HorizontalRaster (List[KeyValue]: Liste aus Key-/Value-Paaren) Beschreibung des horizontalen Layout-Rasters. Vgl. *Raster-bezogene Widget-Properties*.

Beispiel: `[label_links: min, spc1: 6, bar: max, spc2: 6, label_rechts: 64]`

`HorizontalRaster` muss für horizontale Balken definiert sein (d.h. für `Orientation left` oder `right`).

Default: `chart: max`

VerticalRaster (List[KeyValue]: Liste aus Key-/Value-Paaren) Beschreibung des vertikalen Layout-Rasters. Vgl. "Raster".

Beispiel: `[label_oben: min, spc1: 6, bar: max, spc2: 6, label_unten: 64]`

`VerticalRaster` muss für vertikale Balken definiert sein (d.h. für `Orientation up` oder `down`).

Default: `main: max`

RasterDebugColor (Color) Wenn gesetzt, wird ein Debug-Raster der Rasterpositionen gezeichnet.

Values (Table) Die Tabelle bzw. Liste der darzustellenden Werte. Values wandelt eine übergenene Liste selbstätig in eine geeignete Tabelle. Zweidimensionale Daten können direkt als Tabelle oder als List[List] zugewiesen werden.

Direkte Zuweisung mit List[Number]:

```
<property name="Values" type="List[Number]" value="1,2,3,4" />
```

Binding an den Controller (vgl. ColumnNames). Die Property `Values` erhält so die TabellenSpalte mit dem Namen "percent".:

```
<bind property="Values" to="Controller.Columns" key="percent" />
```

Colors (List[Color]) Die Farben, die das Chart für die einzelnen Werte nutzen soll. Hiermit werden dann Balken, Säulen, Kreissegmente oder Linien eingefärbt. Wenn mehr Elemente als Farben vorhanden sind,

Direkte Zuweisung mit List[Color]:

```
<property name="Colors" type="List[Color]" value="#aaa,#999,#888,#777" />
```

Binding an den Controller (vgl. ColumnNames). Die Property `Colors` erwartet die gesamte Liste der Werte:

```
<bind property="Colors" to="Controller.Columns" key="color" />
```

Groups (List[String]) Die Liste der Gruppen, die bei der Darstellung der Werte genutzt werden sollen:

```
<property name="Groups" type="List[String]" value="Gruppe1,,Gruppe2," />
<property name="Groups" type="List[String]" value="Gruppe1,Gruppe1,Gruppe2,Gruppe2" />
```

Beide Schreibweisen sind gleichbedeutend, d.h. es reicht aus, das erste Element einer Gruppe mit einem Gruppennamen zu versehen, es ist aber auch möglich, alle Elemente auszuzeichnen.

Hat die Liste nicht dieselbe Größe wie `Values`, so werden überzählige Elemente ignoriert bzw. benötigte weitere Elemente mit leeren Strings `""` gefüllt. Ein fünfter Wert würde bei obenstehender `Groups`-Definition also der Gruppe2 angehören.

Binding an den Controller (vgl. ColumnNames):

```
<bind property="Groups" to="Controller.Columns" key="groups" />
```

Minimum (Number)

Default: 0

Maximum (Number)

Default: 100

AutoScale (Symbol) Bei `AutoScale=detect` werden die Balken des Charts automatisch anhand der tatsächlichen Werte skaliert. Bei `AutoScale=none` werden stattdessen die Werte in Minimum und Maximum genutzt.

Werte: `none, minimum, maximum, both, equal, detect`

Default: `detect`

AutoScale	nutzt Daten min	nutzt Daten max	nutzt Minimum	nutzt Maximum	pos+neg gleich
none			X	X	
minimum	X			X	
maximum		X	X		
both	X	X			
equal	X	X			X
detect	X	X			X (wenn min < 0 und max > 0)

Section author: jo

Note: (jo)

Bitte konkretisieren (mao)

LayoutConstraints sh. ChildMover (ChildMover wertet die LayoutConstraints des Widgets aus) Das ist relativ komplex und undurchsichtig (insbesondere AlignMode). Außerdem kommt es mir so vor, dass bei AlignMode = `ignore` zu viele Dinge abgeklemmt werden, nicht nur das abschließende Alignen.

In ChildMover werden noch Widgets removed und neu added.

Stand nach Class-Inspection 31.5.2010. ChildMover heißt da noch ChildMover2.

Überprüfen: Die Angaben zu Anchor!

7.3 Raster und LayoutConstraints

Das Chartlayout mithilfe eines Rasters ist ein mehrschrittiges Verfahren, in dem die Chartelemente (gemeint sind im Allgemeinen Chart-Labels) zunächst in eine vordefinierte Rasterposition layoutet werden, um dann in einem zweiten Layout-Schritt noch automatisch verschoben zu werden, beispielsweise um eine spezifische Ausrichtung zu erreichen.

Der MAD-Autor definiert zunächst ein `HorizontalRaster` und/oder ein `VerticalRaster` und kann dabei neben fixen Breiten-/Höhenangaben auch `min` und `max` als Auto-Funktionen nutzen.

Die so definierten Rasterpositionen können dann in den LabelTemplates als `LayoutConstraints` benutzt werden.

Dieses Standard-Verhalten lässt sich modifizieren, in dem für X, Y, Breite und Höhe abweichende `...Source`-Angaben gemacht werden und indem die Ausrichtung genauer bestimmt wird.

7.3.1 Raster-bezogene Widget-Properties

HorizontalRaster (*List(KeyValue)*: Liste aus Key-/Value-Paaren) Beschreibung des horizontalen Layout-Rasters. Vgl. "Raster".

Beispiel: `[label1: min; spc1: 6; bar: max; spc2: 6; label2: 64]`

`HorizontalRaster` muss für horizontale Balken definiert sein (d.h. für `Orientation left` oder `right`).

Default: `chart: max`

VerticalRaster (*List(KeyValue)*: Liste aus Key-/Value-Paaren) Beschreibung des vertikalen Layout-Rasters. Vgl. "Raster".

`VerticalRaster` muss für vertikale Balken definiert sein (d.h. für `Orientation up` oder `down`).

Default: `chart: max`

Beschreibung des horizontalen bzw. vertikalen Layout-Rasters:

Die Keys können in der Beschreibung der Chartelemente als `Rasterposition` genutzt werden. Values sind Pixelangaben oder Symbole für spezielle Funktionen (s.u. "Wertangaben"):

```
<property name="VerticalRaster" type="List[KeyValue]" value="[left: 10, chart: max, right: 10]" />
<property name="HorizontalRaster" type="List[KeyValue]" value="[chart: max, label1: 25, label2 :25]"
```

LayoutConstraints (Map)

Sammlung von Layoutvorgaben für ein Chartelement (z.B. Label, Balken, etc.), vgl. unten LayoutConstraints-Angaben.

LayoutConstraints werden auf dem zu layoutenden Element (beliebiges Widget), nicht auf dem Chartwidget angegeben.

Das folgende Beispiel nutzt das (Standard-)Symbol `chart` für den zentralen Chartbereich des Rasters (in `chart` wird das Balken-, Linien- oder Kreischart platziert, wenn in den LayoutConstraints nicht anderes vorgegeben ist.) Im horizontalen Raster sind zwei Label-Positionen definiert, auf die mit LayoutConstraints z.B. so verwiesen werden kann:

```
<property name="VerticalRaster" type="List[KeyValue]" value="[left: 10, chart: max, right: 10]" />
<property name="HorizontalRaster" type="List[KeyValue]" value="[chart: max, label1: 25, label2 :25]"

<template type="Text" name="Wertlabel2">
        <property name="LayoutConstraints">
                <map>
```

```
                        <property name="HorizontalRasterPosition" value="label1" />
                </map>
        </property>
</template>

<!-- oder Kurzschreibweise: -->
<template type="Text" name="Wertlabel1">
        <property name="LayoutConstraints" value="label1;chart"/>
</template>
```

7.3.2 Wert-Angaben in der Raster-Definition

Bei der Definition eines Layout-Rasters können als Werte Pixelangaben, aber auch Symbole zur Festlegung eines automatischen Layouts verwendet werden.

n (Number) Pixelangabe für Höhe/Breite

min Das enthaltene Element sperrt das Raster auf; die Rasterzeile/-spalte wird so breit/hoch wie von `PreferredBounds` des breitesten/höchsten enthaltenen Elements vorgegeben.

max Die Raster-Position wird so breit/hoch wie möglich, d.h. sie nutzt den gesamten zur Verfügung stehenden Platz, nachdem evtl. die Größe weiterer Rasterpositionen bestimmt wurden.

7.3.3 LayoutConstraints-Angaben

LayoutConstraints werden im Template eines Chartelements (=Labels) gesetzt, um die Position und Größe des Elements einzustellen. Bei Verwendung eines Rasters müssen `HorizontalRasterPosition` und `VerticalRasterPosition` gesetzt werden, um die Platzierung des Chartelements zu steuern.

Die weiteren Properties sind nur notwendig, wenn das Layoutverhalten vom Standard (Platzierung und Größe werden durch das Raster vorgegeben) abweichen soll.

HorizontalRasterPosition (Symbol) Gibt an, in welchem horizontalen Rasterfeld die Instanzen des Chartelements gezeichnet werden sollen. Ist kein Symbol für eine Rasterposition angegeben, so werden die Chartelemente an die Position (0,0) des Parentwidgets platziert.

Werte: Alle Symbole, in der Raster-Definition verwendet werden. Wird ein anderes Symbol verwendet, so wird das platzierte Element nicht sichtbar sein.

Default: Point(0,0)

VerticalRasterPosition (Symbol) Gibt an, in welchem vertikalen Rasterfeld die Instanzen des Chartelements gezeichnet werden sollen. Ist kein Symbol für eine Rasterposition angegeben, so werden die Chartelemente an die Position (0,0) des Parentwidgets platziert.

Werte: Alle Symbole, in der Raster-Definition verwendet werden. Wird ein anderes Symbol verwendet, so wird das platzierte Element nicht sichtbar sein.

Default: Point(0,0)

XSource (Symbol) Gibt an, welche Eigenschaft genutzt werden soll, um die X-Position des Chartelements festzulegen. `raster` sorgt dafür, dass der durch das Raster vorgegebene Wert genutzt wird. `anchor` bindet die Elementeigenschaft an die Position des zugehörigen Balkens, der zugehörigen Linie etc... `preferred` und `keep` befragen das Template-Widget nach dessen bevorzugten oder tatsächlichen Wert.

Werte: `raster`, `anchor`, `preferred`, `keep` *?*

Default: `raster`

YSource (Symbol) Gibt an, welche Eigenschaft genutzt werden soll, um die Y-Position des Chartelements festzulegen.

Werte: `raster`, `anchor`, `preferred`, `keep` *?*

Default: `raster`

WidthSource (Symbol) Gibt an, welche Eigenschaft genutzt werden soll, um die Breite des Chartelements festzulegen.

Werte: `raster`, `anchor`, `preferred`, `keep`

Default: `raster`

HeightSource (Symbol) Gibt an, welche Eigenschaft genutzt werden soll, um die Y-Position des Chartelements festzulegen.

Werte: `raster`, `anchor`, `preferred`, `keep`

Default: `raster`

AlignMode (Symbol) Gibt an, ob die Position des Chartelements durch die Alignment-Property zusätzlich modifiziert werden soll (Default: `ignore`, also keine Veränderung). `xy` (deprecated) und `anchor` sind synonym und nutzen als Bezugspunkt die Position des Chartelements, nachdem es platziert wurde; `raster` nutzt die Raster-Position unter Einbeziehung von `PadAlign`.

Werte: `anchor`, `raster`, `ignore`, `preserve`

Default: `ignore`

Alignment (Symbol) Gibt an, wie das Chartelement ausgerichtet werden soll. Der Bezugspunkt der Verschiebung wird in AlignMode angegeben.

Werte:

```
top-left      top      top-right
left          center   right
bottom-left   bottom   bottom-right
```

Default: `top-left` *Im Code: null, sollte aber einen Default haben*

Anchor (Symbol) Angabe, welcher Anchorpunkt genutzt werden soll. Die möglichen Angaben variieren je nach Charttyp.

Was sind die Defaults, wenn Anchor nicht gesetzt wird?

LineChart: grid-x, grid-y, sample-x, sample-y ?

Was bedeuten die Werte bei BarChart?

Werte:

```
LineChart (LineAnchor): line-first, line-second, line-last, line-selected

BarChart (BarWidget):
animation-red, animation-center, animation-green, animation,
half-red, half-center, half-green, half,
base-red, base-center, base-green, base,
maxvalue-red, maxvalue-center, maxvalue-green, maxvalue,
value-red, value-center, value-green, value,
dzero-red, dzero-center, dzero-green, dzero,
front-red, front-center, front-green, front,
zero-red, zero-center, zero-green, zero,
rear-red, rear-center, rear-green, rear
```

```
PieChart: *NYI*
```

7.3.4 Zugriff auf Rasterkoordinaten und -größen

geht das? Die tatsächlichen Pixelwerte für Position und Größe der definierten Rasterelemente können wie folgt ausgelesen werden:

VerticalRaster.<nameRasterElem>.height die Höhe des Rasterelementes <nameRasterElem>

VerticalRaster.<nameRasterElem>.y die obere Position des Rasterelementes <nameRasterElem>

HorizontalRaster.<nameRasterElem>.width die Breite des Rasterelementes <nameRasterElem>

HorizontalRaster.<nameRasterElem>.x die linke Position des Rasterelementes <nameRasterElem>

Section author: jo, benjamin

7.4 Controller und Binding im Chart

7.4.1 Properties

Das Chart hat eine automatisch initialisierte Property `Controller`.

Controller

Magie... Bei Initialisierung des BarCharts wird der `Controller`-Property automatisch ein (leeres) ChartController-Objekt zugewiesen. Diesem kann in einer Action eine Datentabelle zugewiesen werden:

```
<action trigger="data" arguments="data">
  this.Controller.Table = data.Table;
</action>
```

Sodann stehen die Spalten der Datentabelle für Binding zur Verfügung:

```
<bind property="Values" to="Controller" key="lists[2]" />
```

oder, wenn die ColumnNames gesetzt wurden:

```
<bind property="Values" to="Controller.Columns" key="percent" />
```

ColumnNames

Eigentlich eine Controller-Eigenschaft. Sollte auch dorthin. Dann nur über Skript setzbar.

Erlaubt es, die Spalten einer Data-Tabelle zu benennen, was die Zuweisung von Binding-Keys erleichtert.

Beispiel:

```
<property name="ColumnNames" value="companyName,companyColor,percent"/>
<!-- ... -->
<bind property="Values" to="Controller.Columns" key="percent" />
<!-- ... -->
<template name="BarTemplate">
        <bind property="BackgroundColor" to="ControllerProxy" key="companyColor" />
</template>
```

7.4.2 Methoden

setTable(Data.Table) Übergabewert ist ein Data.Table.

setColumnNames(List) Übergabewert ist eine Liste von Strings, die die Spalten der tabellarischen Daten benennt.

setContent(???) Übergabewert ist ein JavaScript-Objekt, dessen Keys als ColumnNames genutzt werden.

```
var temp = new Object();
temp.companyName  = ['Hans', 'Dieter', 'Peter', 'Willi'];
temp.companyColor = ['red', 'green', 'blue', 'yellow'];
temp.percent      = [1 , 2, 3, 4];
this.controller.setContent (temp);
```

7.4.3 Binding an Objekte des Controllers

Controller.lists

Controller.current

Controller.selected

Controller.first

Controller.last

7.4.4 Binden an Chart-Daten

Needs work

In einem LineChartWidget werden Rasterpositionen per Namen in den Eigenschaften *VerticalRaster* und *Horizontal-Raster* definiert.

currentX Diese Eigenschaft ermöglicht das Setzen von Widgets an die horizontalen Datenpositionen.

current.first.x, current.first.y current ist die aktuell gezeichnete Linie. First ist der erste Datenpunkt. x ist die horizontale Position.

current.last.x, *current.last.y current ist die aktuell gezeichnete Linie. Last ist der letzte Datenpunkt. x ist die horizontale Position.

Beispiele um Widgets gegen Rasterpositionen zu binden

```
<widget name="TheLineChart" type="LineChart" x="0" y="50" shape="R 400 300">
        <property name="HorizontalRaster" value="a: 50; chart: max; d:20"/>
        <property name="VerticalRaster" value="v: 25; chart: max; z: 40"/>
</widget>
<template name="zzzXAxisLabels" shape="R 1000,1000" >
        <property name="BackgroundColor" value="transparent"/>
        <widget type="Alignment">
                <property name="BackgroundColor" value="yellow"/>
                <property name="AlignmentXAxis" value="center"/>
                <bind property="AlignedY" to="TheLineChart" key="current.last.y"/>
                <bind property="AlignedX" to="TheLineChart" key="HorizontalRaster.d.x"/> -->
                <widget type="Text" shape="R 40 20">
                        <property name="BackgroundColor" value="blue"/>
                        <bind property="Text" to="TheLineChart" key="currentX"/>
                </widget>
```

```
        </widget>
</template>
```

Section author: jo

7.5 BarChartWidget

Das BarChart-Widget dient dazu, horizontale und waagrechte Balken-/Säulencharts inklusive ihrer Labels zu zeichnen, zu animieren und interaktiv zu gestalten.

Es bietet eine Vielzahl von Einstellmöglichkeiten, mit denen es an den jeweiligen Anwendungszweck angepasst werden kann.

Note: Als einfachsten Fall hätte ich eigentlich lieber so etwas (ohne Data), ohne Raster, mit einem Default-Element-Spacing: (jo)

```
Beispiel 1:
<widget name="Chart0" type="BarChart" x="20" y="20" shape="R 400 300">
  <property name="Orientation" value="up"/>
  <!-- property name="ElementMetrics" value="spacing: 10%"/ DEFAULT!!! -->

  <action trigger="load">
    var temp = new Object();
    temp.companyName  = ['Hans', 'Dieter', 'Peter', 'Willi'];
    temp.companyColor = ['red', 'green', 'blue', 'yellow'];
    temp.percent      = [1 , 2, 3, 4];
    this.controller.setContent (temp);
    this.Values = temp.percent;
  </action>

  <property name="BarTemplates" value="BarTemplate" />
  <template name="BarTemplate">
    <bind property="BackgroundColor"
    to="Controller.current" key="companyColor" />
  </template>
</widget>
```

Note: Stand heute ist:

Dasselbe einfache Säulenchart, das ein Data nutzt, das auf das Chart geroutet wurde:

```
Beispiel 2:
<?xml version="1.0" encoding="UTF-8"?>
<presentation width="340" height="200">

  <include src="../../data/simpledata.inc.mad" />

  <widget name="TheChart" type="BarChart" x="20" y="20" shape="R 300 160">
    <bind property="Data" to="presentation" key="TheData"/>
    <bind property="Values" to="Controller" key="lists.percent" />

    <property name="VerticalRaster" value="chart:max" />
    <property name="ElementMetrics" value="spacing: 10%"/>

    <property name="ColumnNames" value="name;color;percent;diff;group;image"/>
    <action trigger="data" arguments="data">
```

```
      this.Controller.setTableContent(data.get("Table"));
    </action>

    <property name="BarTemplates" value="BarTemplate" />
    <template name="BarTemplate">
      <bind property="BackgroundColor" to="Controller.current" key="color" />
    </template>
  </widget>
</presentation>
```

Die Property `Orientation` beschreibt die Richtung, in die der Balken wachsen soll. `Orientation=up` ist demnach ein Säulenchart. In `VerticalRaster` ist nur ein einziger Layoutbereich mit dem Namen `chart` definiert, der den zur Verfügung stehenden Platz ausfüllen soll (`max`). *Diese beiden Angaben entsprechen den Defaults.* Die einzelnen Elemente (=Balken) sollen mit einem Abstand von 10% (des theoretisch für ein Element zur Verfügung stehenden Platzes) gezeichnet werden.

Das Chart nutzt ein Template für den Balken; die Farbe des Balkens ist (im Template `BarTemplate`) an ein Datenfeld im `Controller` gebunden. In der Property `LayoutConstraints` können für das BalkenTemplate weitere Vorgaben gemacht werden: Hier wird die `VerticalRasterPosition` festgelegt.

7.5.1 Properties

Neben den Properties, die für alle Charts gelten (siehe Abschnitt *Gemeinsame Properties aller Charttypen*), hat das Barchart folgende:

Orientation (Symbol) Die Richtung, in die die Balken gezeichnet werden.

Werte: `up, down, left, right`

Default: `up`

ChartAlign (Symbol) Die Position, an der die Balken oder Säulen innerhalb des Charts gezeichnet werden.

Werte: `top, bottom, left, right, center`

Default: `center`

ElementMetrics (*List(KeyValue)*: Liste aus Key-/Value-Paaren) Erweiterbares Objekt zur Beschreibung von Maßangaben für Chartelemente. Als ein Chartelement wird hier die zu einem Wert gehörige Kombination aus Balken und Labels aufgefasst. Möglich sind die Angaben `before`, `after` und `spacing`, die einen Bereich vor bzw. nach dem Element oder den Zwischenraum zwischen zwei Elementen beschreiben (als Pixel oder Prozentangabe).

Innerhalb der Elemente können die

Ein Chart mit einer ersten Gruppe G1 mit zwei Elementen E1 und E2 und einer zweiten Gruppe G2 mit drei Elementen E3, E4 und E5 hat demnach folgende Struktur (b: before, a: after, s: spacing):

```
bbb_____________________________Chart_____________________________aaa
   bb_________G1________aassbb_____________G2____________aa
     b__E1__asb__E2__a       b__E3__asb__E4__asb__E5__a
```

Spacing kann nur zwischen Elementen in derselben Gruppe stattfinden: zwischen E2 und E3 gibt es kein Element-Spacing.

Default: `spacing: 10%;before: 0; after: 0`

GroupMetrics (*List(KeyValue)*: Liste aus Key-/Value-Paaren) Erweiterbares Objekt zur Beschreibung von Maßangaben für (Werte-)Gruppen. GroupMetrics funktionieren wie ElementMetrics.

Default: `spacing:10%;before: 0; after: 0`

ChartMetrics (*List(KeyValue)*: Liste aus Key-/Value-Paaren) Erweiterbares Objekt zur Beschreibung von Maßangaben für das gesamte Chart. Implementiert sind `before` und `after`, die die äußeren Abstände vor- und nach dem Chart, jeweils von links (beim Säulenchart) bzw. oben (beim Balkenchart) gesehen, als Pixel- oder Prozentangabe beschreiben.

Default: `before: 0; after: 0`

BarMetrics

ColumnNames (List) Dient dazu, die Spalten einer Data-Tabelle im `Controller` zu benennen, was die Zuweisung von Binding-Keys erleichtert. Vgl. "Controller und Binding im Chart".

Beispiel:

```
<property name="ColumnNames" value="companyName;companyColor;percent"/>
```

Controller (ChartController) Objekt des Charts, das Daten verwaltet und als Binding-Target dienen kann. Vgl. "Controller und Binding im Chart".

Default: ein leeres `ChartController`-Objekt.

Nutzen Sie eine JavaScript-Aktion, um Daten zuzuweisen (z.B. mit Controller.setTableContent()), oder weisen Sie der Property Controller den Controller eines andeen Widgets zu, um Daten gemeinsam zu nutzen.

Note: ChartElements fallen weg.

Ersatz: ElementLabels (, SubElementLabels, GroupLabels), BarTemplates

ElementLabels (List(Strings)) Liste von Templatenamen, die als Labels für Elemente verwendet werden sollen.

Default: leere Liste

BarLabels (List(Strings)) Liste von Templatenamen, die als Labels für SubElemente verwendet werden sollen.

Default: leere Liste

GroupLabels (List(Strings)) Liste von Templatenamen, die als Labels für Gruppen verwendet werden sollen.

Default: leere Liste

BarTemplates (List(Strings)) Liste von Templatenamen, die für die Balken verwendet werden sollen.

MinNumberOfBars (Integer) Mit der Angabe von `MinNumberOfBars` kann das Chartlayout modifiziert werden: Die vorhandenen Elemente werden so layoutet, *als ob* es die angegebene Zahl von Elementen gäbe.

Default: 0

MaxNumberOfBars (Integer) Mit der Angabe von `MaxNumberOfBars` werden maximal die angegebene Zahl Balken gezeichnet, d.h. die Werteliste wird für die Darstellung abgeschnitten.

Default: 12

ZeroLineWidth (Number) Beschreibt die Liniendicke der Nullinie.

Default: 0

ZeroLineMode (Symbol) *NYI (war auch in Core nicht implementiert?)*

ZeroLineOffset (Number) Abstand der Nulllinie vom Balken (in pos. und neg. Richtung)

Default: 0

7.5.2 Methoden

???

7.5.3 Signale

???

7.5.4 Beispiele

... sollten etwas komplexer sein also oben! ...

Beispiel für ein Säulenchart mit Groups und Group-Labels. Hier sehen wir drei Charts, bei denen die Group-Labels an den drei möglichen Positionen stehen: body (über den Bars), header (im Header-Bereich) und groups (über beide Bereiche)

Beispiel:

```
<presentation width="1050" height="680">

<property name="Title" value="Simple Bars" />

<template name="GLab" shape="R 10 10">
        <property name="BackgroundColor" value="red"/>

        <property name="LayoutConstraints">
                <map>
                        <property name="HorizontalRasterPosition" value="body"/><!-- body: c
                        <property name="VerticalRasterPosition" value="grp"/>
                        <property name="XSource" value="raster"/>
                        <property name="YSource" value="raster"/>
                        <property name="WidthSource" value="raster"/>
                        <property name="HeightSource" value="keep"/>
                        <property name="AlignMode" value="raster"/>
                        <property name="Align" value="top-left"/>
                </map>
        </property>

</template>

<template name="GLab2" shape="R 10 10">
        <property name="BackgroundColor" value="red"/>

        <property name="LayoutConstraints">
                <map>
                        <property name="HorizontalRasterPosition" value="header"/><!-- body:
                        <property name="VerticalRasterPosition" value="grp"/>
                        <property name="XSource" value="raster"/>
                        <property name="YSource" value="raster"/>
                        <property name="WidthSource" value="raster"/>
                        <property name="HeightSource" value="keep"/>
                        <property name="AlignMode" value="raster"/>
                        <property name="Align" value="top-left"/>
                </map>
        </property>
```

```
</template>

<template name="GLab3" shape="R 10 10">
        <property name="BackgroundColor" value="red"/>

        <property name="LayoutConstraints">
                <map>
                        <property name="HorizontalRasterPosition" value="group"/><!-- body:
                        <property name="VerticalRasterPosition" value="grp"/>
                        <property name="XSource" value="raster"/>
                        <property name="YSource" value="raster"/>
                        <property name="WidthSource" value="raster"/>
                        <property name="HeightSource" value="keep"/>
                        <property name="AlignMode" value="raster"/>
                        <property name="Align" value="top-left"/>
                </map>
        </property>

</template>


<widget name="TheChart" type="BarChart" x="20" y="20" shape="R 300 300">
        <property name="Debug" value="false"/>
        <property name="BorderWidth" value="1"/>
        <property name="VerticalRaster" type="List[KeyValue]" value="grp:40, chart: max" />
        <property name="Values" type="List[Number]" value="1,2,3,4,5,6,7"/>
        <property name="GroupMetrics" value="spacing: 20,header: false"/>
        <property name="Orientation" value="up"/>
        <property name="Groups" type="List[String]" value="A,,,B,,,"/>

        <property name="GroupLabels" value="[@GLab]"/>

</widget>

<widget name="TheChart2" type="BarChart" x="360" y="20" shape="R 300 300">
        <property name="Debug" value="false"/>
        <property name="BorderWidth" value="1"/>
        <property name="VerticalRaster" type="List[KeyValue]" value="grp:40, chart: max" />
        <property name="Values" type="List[Number]" value="1,2,3,4,5,6,7"/>
        <property name="GroupMetrics" value="spacing: 20,header: true"/>
        <property name="Orientation" value="up"/>
        <property name="Groups" type="List[String]" value="A,,,B,,,"/>

        <property name="GroupLabels" value="[@GLab2]"/>

</widget>

<widget name="TheChart3" type="BarChart" x="700" y="20" shape="R 300 300">
        <property name="Debug" value="false"/>
        <property name="BorderWidth" value="1"/>
        <property name="VerticalRaster" type="List[KeyValue]" value="grp:40, chart: max" />
        <property name="Values" type="List[Number]" value="1,2,3,4,5,6,7"/>
        <property name="GroupMetrics" value="spacing: 20,header: true"/>
        <property name="Orientation" value="up"/>
        <property name="Groups" type="List[String]" value="A,,,B,,,"/>

        <property name="GroupLabels" value="[@GLab3]"/>
```

```
</widget>

<widget type="Text" x="50" y="350" shape="R 80 22">
        <property name="Text" value="Header"/>
</widget>

<widget type="Text" x="120" y="350" shape="R 30 22">
        <property name="BorderWidth" value="1"/>
        <property name="Padding" value="2"/>
        <property name="Interactive" value="true"/>
        <property name="Text">on</property>
        <action trigger="pointer-up">
                this.parent.TheChart.GroupMetrics = $$("List[KeyValue]", "spacing: 20,header
                this.parent.TheChart2.GroupMetrics = $$("List[KeyValue]", "spacing: 20,heade
                this.parent.TheChart3.GroupMetrics = $$("List[KeyValue]", "spacing: 20,heade
        </action>
</widget>

<widget type="Text" x="170" y="350" shape="R 30 22">
        <property name="BorderWidth" value="1"/>
        <property name="Padding" value="2"/>
        <property name="Interactive" value="true"/>
        <property name="Text">off</property>
        <action trigger="pointer-up">
                this.parent.TheChart.GroupMetrics = $$("List[KeyValue]", "spacing: 20,header
                this.parent.TheChart2.GroupMetrics = $$("List[KeyValue]", "spacing: 20,heade
                this.parent.TheChart3.GroupMetrics = $$("List[KeyValue]", "spacing: 20,heade
        </action>
</widget>


<widget type="Text" x="50" y="380" shape="R 80 22">
        <property name="Text" value="Groups"/>
</widget>

<widget type="Text" x="120" y="380" shape="R 60 22">
        <property name="BorderWidth" value="1"/>
        <property name="Padding" value="2"/>
        <property name="Interactive" value="true"/>
        <property name="Text">2 Groups</property>
        <action trigger="pointer-up">
                this.parent.TheChart.Groups = $$("List[String]", "A,,,B,,,");
                this.parent.TheChart2.Groups = $$("List[String]", "A,,,B,,,");
                this.parent.TheChart3.Groups = $$("List[String]", "A,,,B,,,");
        </action>
</widget>

<widget type="Text" x="190" y="380" shape="R 60 22">
        <property name="BorderWidth" value="1"/>
        <property name="Padding" value="2"/>
        <property name="Interactive" value="true"/>
        <property name="Text">3 Groups</property>
        <action trigger="pointer-up">
                this.parent.TheChart.Groups = $$("List[String]", "A,,B,,C,");
                this.parent.TheChart2.Groups = $$("List[String]", "A,,B,,C,");
                this.parent.TheChart3.Groups = $$("List[String]", "A,,B,,C,");
        </action>
</widget>
```

```
        <widget type="Text" x="260" y="380" shape="R 60 22">
                <property name="BorderWidth" value="1"/>
                <property name="Padding" value="2"/>
                <property name="Interactive" value="true"/>
                <property name="Text">0 Groups</property>
                <action trigger="pointer-up">
                        this.parent.TheChart.Groups = null;
                        this.parent.TheChart2.Groups = null;
                        this.parent.TheChart3.Groups = null;
                </action>
        </widget>
```

</presentation>

Section author: malte/jo

Note: XXXXXXXXXXXXXXX Status 18.10.2010: verbindlich. XXXXXXXXXXXXXXXXXXXX

Grid: Kann in einen Abschnitt Gridwidget verschoben werden, wenn andere Charttypen das auch nutzen.

7.6 LineChartWidget @

Das LineChart-Widget dient dazu Linien- oder Punkte-Charts inklusive ihrer Labels zu zeichnen, zu animieren und interaktiv zu gestalten.

7.6.1 Beispiele

Einfaches Beispiel eines LineCharts (1 Linie) ohne Labels:

```
Beispiel 1

<widget name="TheLineChart" type="LineChart" x="10" y="10" shape="R 500 400">

  <!--  eindimensionale werte für eine linie -->
  <property name="Values" type="List[Number]" value="[[1,3,5,7,4,2],[2,3,4,6,9,12]]"/>
  <property name="Colors" type="List[Color]" value="[red,green,blue]"/>
</widget>
```

Einfaches Beispiel mit mehreren Linien:

```
Beispiel 2

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <define name="Line1" type="List[Number]" value="[1,3,5]" />
  <define name="Line2" type="List[Number]" value="[3,5,2]" />
  <property name="Values" type="List" value="[@Line1,@Line2]"/>
  <property name="Colors" type="List[Color]" value="red,blue"/>
</widget>
```

7.6.2 Daten & Darstellung

Das LineChartWidget stellt eine oder mehrere Linien dar. Jede Linie besteht aus n zu zeichnenden Werten.

Wird "SamplePositions" nicht gesetzt, so geht die Widget-Logik davon aus, dass die Werteangaben in gleichem Abstand linear hintereinander gezeichnet werden sollen. Dieses Verhalten kann durch Samplepositions beeinflusst werden; liegen beispielsweise Werte für die Datumsangaben 1.3.2010, 1.4.2010 und 1.6.2010 vor, die mit den entsprechenden Abständen auf einer Datumsachse angeordnet werden sollen, so lässt sich das mit der Angabe in `SamplePositions` erreichen:

```
Beispiel 3

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <define name="Line1" type="List[Number]" value="[1,3,5]" />
  <define name="Line2" type="List[Number]" value="[3,5,2]" />
  <property name="Values" type="List" value="[@Line1,@Line2]"/>
  <property name="Colors" type="List[Color]" value="[red,blue]"/>
  <property name="SamplePositions" type="List[Date]" value="[2010-03-01,2010-04-01,2010-06-01]" />
</widget>
```

Damit sind die ersten Elemente der beiden Wertelisten dem Datum 2010-03-01 zugeordnet, die zweiten dem Datum 2010-04-01 und die dritten dem Datum 2010-06-01. Im LineChart wird der Abstand zwischen erstem und zweiten Datenpunkt kleiner Sein als der zwischen zweiten und drittem.

SamplePositions können auch für jede Linie getrennt angegeben werden. Hier wird für die zweite Linie auch ein Wert für den 1. Mai angegeben:

```
Beispiel 4

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <define name="Line1" type="List[Number]" value="[1;3;5]" />
  <define name="Line2" type="List[Number]" value="[3;5;2]" />
  <define name="SamplePositions1" type="List[Date]" value="[2010-03-01;2010-04-01;2010-06-01]" />
  <define name="SamplePositions2" type="List[Date]" value="[2010-03-01;2010-04-01;2010-05-01;2010-06-

  <property name="Values" type="List" value="[@Line1;@Line2]"/>
  <property name="Colors" type="List[Colors]" value="[red,blue]"/>
  <property name="SamplePositions" type="List[List]" value="[@SamplePositions1;@SamplePositions2]"/>
</widget>
```

Die Angaben in `Minimum` und `Maximum` beziehen sich auf die Skalierung der Werte in Y-Richtung; wie in den anderen Charttypen bieten sie auch im Linechart die Möglichkeit, den darzustellenden Wertebereich anzugeben. Liegen Werte außerhalb dieses Bereichs, so werden sie nicht dargestellt, wenn die entsprechende Einstellung der Property `AutoScale` gewählt wird.

Um aus der Menge der Samples nur einen Ausschnitt anzuzeigen, gibt es für die X-Achse einen analogen Mechanismus: Mit `XAxisMinimum`, `XAxisMaximum` und der Angabe `none` für `XAxisAutoScale` lässt sich der Darstellungsbereich in Richtung der X-Achse festlegen. `XAxisMinimum` und `XAxisMaximum` erwarten dabei SamplePositions als Werte (also Datumsangaben oder den Index des jeweiligen Samples):

```
Beispiel 5

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <!-- eine Linie -->
  <property name="Values" type="List[Number]" value="[1;3;5;7;4;2]"/>
  <property name="Colors" type="List[Colors]" value="[red]"/>
  <property name="XAxisMinimum" type="Number" value="1" />
  <property name="XAxisMaximum" type="Number" value="3" />
  <property name="XAxisAutoScale" type="Symbol" value="none" />
</widget>
```

Dargestellt werden hier nur die erste bis dritte Indexposition der Liste, also die Werte 3, 5 und 7. Der erste Wert hat die Indexposition `0`.

7.6.3 LineWidget nutzen

Um die Darstellung der Linien im LineChart beeinflussen zu können, kann in der Property `LineTemplates` eine Liste von Templates (vom Typ `Line` angegeben werden, die der Reihe nach zum Zeichnen der Linien benutzt werden. Vgl. *LineWidget*. Das folgende Beispiel zeichnet 3 Pixel dicke Linien mit Markierungspunkten (Bobbles). Die Bobbles werden im LineTemplate in der Property `BobbleTemplates` ebenfalls explizit angegeben:

```
Beispiel 6

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <define name="Line1" type="List[Number]" value="1;3;5" />
  <define name="Line2" type="List[Number]" value="3;5;2" />
  <property name="Values" type="List[List]" value="Line1;Line2"/>
  <property name="Colors" type="List[Colors]" value="red,blue"/>
  <property name="SamplePositions" type="List[Number]" value="2010-03-01;2010-04-01;2010-06-01" />

  <property name="LineTemplates" type="List[String]" value="myLineTemplate" />
</widget>

<template name="myLineTemplate" type="Line">
  <bind property="BackgroundColor" to="ControllerProxy" key="color" />
  <property name="LineWidth" value="3"/>
  <property name="DrawBobbles" value="true"/>
  <property name="DrawLine" value="true"/>

  <property name="BobbleTemplates" type="List[String]" value="myBobbleTemplate" />
</template>

<template name="myBobbleTemplate">
  <property name="Shape" value="M -1.5,-6 L 1.5,-6 C 2.5,-6 3.5,-5.0 3.5,-4 L 3.5,4
  C 3.5,5.0 2.5,6 1.5,6 L -1.5,6 C -2.5,6 -3.5,5.0 -3.5,4.5 L -3.5,-4 C -3.5,-5.0 -2.5,-6 -1.5,-6z"/>
  <property name="BackgroundColor" value="white"/>
  <property name="BorderColor" value="black"/>
  <property name="BorderWidth" value="1"/>
</template>
```

7.6.4 Grid

Die Darstellung eines Grids im LineChart wird durch die Definition eines Gridwidgets mit `name="Grid"` innerhalb des Linechart-Widgets ermöglicht:

```
<widget type="LineChart">
  <widget type="Grid" name="Grid" />
  <!-- ... -->
</widget>
```

Das Gridwidget hat eine Reihe eigener Properties; außerdem nutzt es innerhalb des Linecharts auch dessen AutoScale-Einstellungen und - wenn gewünscht - die `XAxisAnchors` und `YAXisAnchors` oder die `SamplePositions`.

Gridbezogene Properties des LineCharts *ToBeDiscussed*

XAxisGridMode (Symbol)

Werte: `auto` (be clever), `none` (kein Grid), `gridwidget` (Einstellungen des GridWidgets nutzen), `anchors` (XAxisAnchors nutzen), `samples` (an jeder Sample-Position eine Linie zeichnen)

Default: `gridwidget`

YAxisGridMode (Symbol)

Werte: `auto` (be clever), `none` (kein Grid), `gridwidget` (Einstellungen des GridWidgets nutzen), `anchors` (YAxisAnchors nutzen), `values` (für jeden Wert eine Linie zeichnen)

Default: `gridwidget`

XAxisGridStep (Number) **later**

YAxisGridStep (Number)

Die Angabe erfolgt in Werteeinheiten. Wenn gesetzt, wird die Property im `YAxisGridMode = "auto"` genutzt, um einen bestimmten Abstand der Gridlinien zu erzwingen.

Properties des GridWidgets

Das GridWidget nutzt die folgenden Properties, um ein einfaches Grid zu zeichnen, wenn das ParentChart nicht die Kontrolle übernimmt.

NumberHorizontalLines (Number)

Anzahl der horizontalen Linien, die das Gridwidget platzieren soll. Wenn gesetzt, wird die Angabe für HStep nicht genutzt.

Default: `0`

HorizontalStep (Number)

Abstand der horizontalen Linien in Pixeln. Wenn gesetzt, wird die Angabe für NumberHLines nicht genutzt.

Default: `0`

NumberVerticalLines Anzahl der vertikalen Linien, die das Gridwidget platzieren soll. Wenn gesetzt, wird die Angabe für VStep nicht genutzt.

Default: `0`

VerticalStep Abstand der vertikalen Linien in Pixeln. Wenn gesetzt, wird die Angabe für NumberHLines nicht genutzt.

Default: `0`

ForegroundColor Das GridWidget nutzt die Widget-Property `ForegroundColor` als Grid-Farbe.

Beispiel:

```
Beispiel 6

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <define name="Line1" type="List[Number]" value="1;3;5" />
  <define name="Line2" type="List[Number]" value="3;5;2;9" />
  <define name="SamplePositions1" type="List[Number]" value="2010-03-01;2010-04-01;2010-06-01" />
  <define name="SamplePositions2" type="List[Number]" value="2010-03-01;2010-04-01;2010-05-01;2010-06

  <property name="Values" type="List[List]" value="Line1;Line2"/>
  <property name="Colors" type="List[Colors]" value="red,blue"/>
  <property name="SamplePositions" type="List[List]" value="SamplePositions1;SamplePositions2"/>
```

```
  <!-- Grid: -->
  <property name="XAxisGridMode" value="samples" />
  <property name="YAxisGridMode" value="auto" />
  <widget type="Grid" name="Grid">
    <property name="ForegroundColor" value="#ffffff/80" />
  </widget>
</widget>
```

Im Beispiel wird ein halbtransparentes Grid gezeichnet; die vertikalen Linien befinden sich an allen Samplepositionen; die horizontalen Linien werden automatisch platziert. Da die Werte zwischen 1 und 9 liegen, sind Gridlinien bei 0, 2, 4, 6, 8 zu erwarten.

Auto-Modus:

```
Werte       Step       Gridlinien
0-10        2          0,2,4,6,... (bis zum höchsten Wert)
0-20        5          0,5,10,...
0-50        10         0,10,20,30,...

etc.. Entsprechend die größeren und kleineren Dimensionen:

0-1         0.2        0.2,0.4,0.6,...
0-100       20         20,40,60,...
0-1000      200        200,400,600,...

Wenn das Chart nicht von 0 ausgehend gezeichnet werden soll (AutoScale=none, Minimum=x):

                                                  Step   Gridlinien
z.B. Werte im Bereich 25 bis 35: Differenz ist 10    2    26,28,30,32,34
z.B. Werte im Bereich 25 bis 36: Differenz ist 11    5    25,30,35
z.B. Werte im Bereich 9 bis 36:  Differenz ist 27   10    10,20,30
z.B. Werte im Bereich 5 bis 40:  Differenz ist 35   10    10,20,30,40

Wenn das Chart einen positiven und negativen Darstellungsbereich hat,
wird als Differenz der größte Absolutwert aller Werte genutzt.

Werte       Step       Gridlinien
-10 - 10    2          ...,-6,-4,-2,0,2,4,6,...
-1 - 5      1          -1,0,1,2,3,4,5 (AutoScale=both oder minimum)
                       -5,-4,-3,-2,-1,0,1,2,3,4,5 (AutoScale=equal oder detect)
```

7.6.5 Interaktivität

Das LineChart kennt drei spezielle Signale: `chart-pointer-down`, `chart-pointer-up` und `chart-pointer-move`. Diese Signale stellen zwei Informationen zur Verfügung:

1. den der Entfernung nach nächsten Datenpunkt über alle Linien basierend auf dem Ursprung des Signals (`nearest`)
2. eine Liste mit einem vertikalen Querschnitt durch alle Linien: basierend auf der X-Position des Signals wird die nächste Sampleposition bestimmt, für die alle zugehörigen Datenpunkte zurückgeliefert werden (`interceptions`)

Dieses Beispiel zeigt eine Action, welche bei Mausbewegungen einen Marker an den nächstliegenden Datenpunkt setzt und den Wertequerschnitt in einem separaten Balkenchart darstellt:

```
Beispiel 7

<action trigger="chart-pointer-move" arguments="nearest,interceptions">
  <![CDATA[

  var ctrl = this.Controller;

  // place marker widget at interceptionpoint
  $(this,"Marker").Location = nearest.selectionPoint;

  // create text for the overlay:
  var t = ctrl.getValue(nearest.lineIndex, "label");
  t = t + " " + ctrl.getValue(nearest.lineIndex, "values").get(nearest.index);
  t = t + "%";

  $(this, "Marker").Text = t;

  // set color in overlay to color from data in nearest datapoint
  $(this, "Marker.FarbMarker").BackgroundColor = ctrl.getValue(nearest.lineIndex, "color");

  // create list with values from all interceptions
  var mylist = Type.List();
  for(var i=0;i<interceptions.size();i++){
    var row = Type.List();
    row.add(ctrl.getValue(i, "label"));
    row.add(ctrl.getValue(i, "color"));
    row.add(ctrl.getValue(i, "values").get(nearest.index));
    mylist.add(row);
  }

  // set the values of the barchart to the generated list of values from the chartevent
  theBarChart.Controller.setContent(mylist);
  ]]>
</action>
<widget type="Text" name="Marker" shape="R 80 50 118 35">
  <property name="Visible" value="false"/>
  <property name="BackgroundColor" value="#ffffff"/>
  <property name="Padding" value="10 10 10 25"/>
  <widget name="FarbMarker"></widget>
</widget>
```

Note: neu schreiben

Label

Es gibt verschiedene Arten von Labels welche im LinechartWidget vorhanden sind.

Line Labels, Start- und Endlabel an Linien

Labels, die für jede Linie erzeugt werden. Üblicherweise werden diese am Beginn bzw. Ende einer Linie platziert und dienen in der Regel dazu den Anfangs- bzw. Endwert einer Linie anzuzeigen. Die Positionierung der Label kann sich am Raster des gesamten LineChartWidgets orientieren oder an einzelnen Ankerpunkten einer Linie.

X-Achsen Labels

Achsenbeschriftungen für die X-Achse. (TBD) *Anchor wie geht das?*

Y-Achsen Labels

Achsenbeschriftungen für die Y-Achse. (TBD) *Anchor wie geht das?*

7.6.6 Properties

Neben den Properties, die für alle Charts gelten (siehe Abschnitt *Gemeinsame Properties aller Charttypen*), hat das LineChart folgende eigene Properties:

HitDistance (Number)

TODO

Default: `0`

Linien:

SamplePositions (Table *tbd*)

Weist die Values Samples zu, vgl. Beispiele 3 und 4.

Default: `null`

LineLabels (List[String])

Liste der Elemente welche pro Linie zur Beschriftung genutzt werden. Über ein Binding auf die momentane Linie im `Controller` kann der Wert der aktuellen Linie als Labelbeschriftung genutzt werden.

Beispiel:

```
Beispiel 8

<template type="Text" name="StartLabel" shape="R 10 10">
  <bind property="Text" to="ControllerProxy" key="values.first"/>
  <!-- hier fehlen LayoutConstraints, die die Position des Labels festlegen -->
  <!-- ... -->
</template>

<template type="Text" name="EndLabel" shape="R 10 10">
  <bind property="Text" to="Controller.current" key="values.last"/>
  <!-- hier fehlen LayoutConstraints, die die Position des Labels festlegen -->
  <!-- ... -->
</template>

<widget type="LineChart">
  <property name="PerLineElements" value="StartLabel;EndLabel"/>
  <!-- ... -->
</widget>
```

Hier fehlt das Raster, das anzeigt, wo die Labels platziert sind Das erste Objekt in der Liste wird als Label am Start der Linie platziert, das zweite Objekt als Label am Endpunkt der Linie. Als Werte sind die jeweiligen Werte der Linie gebunden.

X-Achse

XAxisAnchors (List[Number])

Liste mit Werten, an denen die Labels der X-Achse platziert werden. Mit dieser Property werden die X-Achsenbeschriftungen eingeschaltet. Wenn XAxisAnchors nicht gesetzt ist, weden auch keine Labels generiert.

Beispiel:

```
<property name="XAxisPositions" value="0;10;20;30;40;50"/>
```

Default: `null`

XAxisAnchorTexts (List[String])

Liste mit Beschriftungen, die für die Labels der X-Achse genutzt werden.

Beispiel:

```
<property name="XAxisAnchorText" value="Label 1,Label 2,Label 3" type="List[String]"/>
```

Default: `null`

XAxisLabels (List[String])

Verweis auf ein oder mehrere Templates, welche für die zu erzeugenden Labels an der X-Achse genutzt werden sollen. In diesen kann über die Property `XAxisController` ein Binding auf Werte der Property `XAxisLabels` erfolgen.

Beispiel:

```
<template type="Text" name="XLabel" shape="R 10 10">
  <bind property="Text" to="ControllerProxy" key="label"/>
   <!-- ... -->
</template>
<!-- ... -->
<widget type="LineChart">
  <property name="XAxisLabels" value="[@Label 1, @Label 2, @Label 3]"/>
  <!-- ... -->
</widget>
```

Default: `null`

XAxisController (ChartController)

Bei Initialisierung des LineCharts wird der `XAxisController`-Property automatisch ein (leeres) ChartController-Objekt zugewiesen. Mittels Binding kann auf Werte des XAxisControllers zugegriffen werden (z.B. XAxisAnchorText).

Beispiel:

```
<bind property="Text" to="ControllerProxy" key="label"/>
```

Default: `leerer Controller`

XAxisMinimum (Number)

Minimalwert auf der X-Achse.

Default: 0

XAxisMaximum (Number)

Maximalwert auf der X-Achse.

Default: 0

XAxisAutoScale (Symbol)

Sofern diese Property auf `true` steht werden die minimalen/maximalen Werte und damit die Skalierung der X-Achse aus den darzustellenden Werten ermittelt. Wenn diese Property auf `false` steht müssen die Properties `MinimumX` , `MaximumX` gesetzt werden um die Skalierung der Achsen festzulegen.

Default: `true`

Y-Achse (angegebene Properties analog X-Achse)

YAxisAnchors (List[Number]) Mit dieser Property werden die Y-Achsenbeschriftungen eingeschaltet. Wenn YAxisAnchors

nicht gesetzt ist, weden auch keine Labels generiert.

YAxisAnchorTexts (List[Object])

YAxisLabels (List[String])

YAxisController (ChartController)

sowie die Widget-Properties, die sich ebenfalls auf die Werte-Dimension, also die Y-Achse, beziehen:

Minimum (Number)

Maximum (Number)

AutoScale (Symbol) `detect` (Default), `none, minimum, maximum, both, equal`

7.6.7 Methoden

???

7.6.8 Signale

chart-pointer-down

Dieses Signal wird ausgelöst, wenn die Maus über einem Chart gedrückt wird.

chart-pointer-up

Dieses Signal wird ausgelöst, wenn die Maus über einem Chart losgelassen wird.

chart-pointer-move

Dieses Signal wird ausgelöst, wenn die Maus über einem Chart bewegt wird.

7.6.9 weitere Beispiele

Beispiel eines LineCharts mit Labels, Rasterpositionen etc.

```
*TODO check*

 <!-- ... -->
 <!-- Template für die Wertebeschriftung auf der Y-Achse -->
 <template name="Dot" shape="R 10 10">
   <property name="BackgroundColor" value="red"/>
   <property name="LayoutConstraints">
     <map type="whatever">
       <!-- Position im Raster horizontal/vertical  siehe Rasterdefinitionen im LineChartWidget -->
```

```
      <property name="HorizontalRasterPosition" value="yla"/>
      <property name="VerticalRasterPosition" value="chart"/>
      <!--  Positionierung in x/y -->
      <property name="XSource" value="raster"/>
      <property name="YSource" value="anchor"/>
      <!--  Größe wie vom Widget gewünscht -->
      <property name="WidthSource" value="preferred"/>
      <property name="HeightSource" value="preferred"/>
      </map>
  </property>

</template>

<template type="Text" name="XLabel" shape="R 10 10">
  <property name="BackgroundColor" value="#8080ff"/>
  <!-- property name="Text" value="-"/-->
  <bind property="Text" to="ControllerProxy" key="label"/>
  <property name="LayoutConstraints">
    <map type="whatever">
      <!-- Position im Raster horizontal/vertical  siehe Rasterdefinitionen im LineChartWidget -->
      <property name="HorizontalRasterPosition" value="chart"/>
      <property name="VerticalRasterPosition" value="xl"/>
      <!--  Positionierung in x/y -->
      <property name="XSource" value="anchor"/>
      <property name="YSource" value="raster"/>
      <!--  Größe wie vom Widget gewünscht -->
      <property name="WidthSource" value="preferred"/>
      <property name="HeightSource" value="preferred"/>
      </map>
  </property>
</template>

<template type="Text" name="StartLabel" shape="R 10 10">
  <property name="Padding" value="4"/>
  <property name="BackgroundColor" value="#80ff80"/>
  <property name="Text" value="-"/>
  <bind property="Text" to="ControllerProxy" key="values.first"/>
  <property name="LayoutConstraints">
    <map type="whatever">
      <property name="Template" value="StartLabel"/>
      <property name="Preset" value="left-of-anchor"/>
      <property name="HorizontalRasterPosition" value="fl"/>
      <property name="XSource" value="raster"/>
      <property name="YSource" value="anchor"/>
      <property name="WidthSource" value="raster"/>
      <property name="HeightSource" value="preferred"/>
      <property name="Anchor" value="line-first"/>
      <property name="AlignMode" value="xy"/>
      <property name="Align" value="right"/>
    </map>
  </property>
</template>

<widget name="TheLineChart" type="LineChart" x="50" y="100" shape="R 500 400">
  <!--  auch andere Widgets können im Raster des LineChart Widgets verankert werden -->
  <widget type="Widget" name="ExampleWidget" shape="R 100 100">
    <property name="LayoutConstraints" type="String" value="chart;xl"/>
  </widget>
```

```
  <!--  was genau macht das hier??? -->
  <widget type="Grid" name="DebugRaster" shape="R 200 200">
    <property name="BoundingBox" type="Shape" value="R 0 0 500 400" />
    <property name="BackgroundColor" value="#000080"/>
  </widget>

  <!-- LineChart reagiert auf Mausbewegungen/Klicks -->
  <property name="Interactive" value="true"/>

  <!-- Aufteilung  -->
  <property name="HorizontalRaster" value="yla: 60; fl: 60; s1: 4; chart: max; ll:60"/>
  <property name="VerticalRaster" value="chart: max; xl: 30; s: 10"/>


  <!--  Benennung der Spalten in der Datentabelle für interne Verwendung -->
  <property name="ColumnNames" value="label;color;values[*]"/>

  <!--  im LineChart sollen die Werte aus der im Controller enthaltenen Liste angezeigt werden -->
  <bind property="Values" to="Controller.Columns" key="values"/>

  <!-- Positionierung und Beschriftung der X-Achse -->
  <property name="XAxisValues" value="0;10;20;30;40;50"/>
  <property name="XAxisLabels" type="List[String]" value="A;B;C;D;E;F"/>

  <!-- Positionierung und Beschriftung der X-Achse -->
  <property name="YAxisValues" value="0;3;7;12;20;35"/>
  <property name="YAxisLabels" type="List[String]" value="A;B;C;D;E;F"/>

  <!-- Template zum Zeichnen der Linie -->
  <template name="LineTemplate" type="Line">
    <property name="BackgroundColor" value="#ff0000"/>
    <property name="LineWidth" value="3"/>
    <property name="DrawBobbles" value="false"/>
    <bind property="BackgroundColor" to="ControllerProxy" key="color"/>
  </template>

  <!--  welche Templates sollen für welche Chartelemente genutzt werden -->
  <property name="XAxisElements" value="XLabel"/>
  <property name="YAxisElements" value="Dot"/>>
  <property name="PerLineElements" value="StartLabel"/>

  <!-- nur fuer das raster -->
  <template name="ElementTemplate">
    <property name="BorderWidth" value="3"/>
    <property name="BorderColor" value="green"/>
    <property name="BackgroundColor" value="transparent"/>
  </template>

  <!--  eintreffende Daten im Controller speichern und dort zur Verfügung stellen -->
  <action trigger="data" arguments="data">
    this.Controller.setTableContent($(data, "Table[0..-1;1..-1][*]"));
  </action>

</widget>
<!-- ... -->
```

Section author: carsten

7.7 PieChartWidget

Das Pie-Chart zeigt Daten als Tortenstücke (Kreissegmente) an. Im Pie-Chart sind keine Labels vorgesehen. Diese müssen ggf. mittels einer LabelBar zusätzlich angezeigt werden.

Innerhalb des Pie-Charts werden Templates zur Erzeugung der einzelnen Segmente genutzt.

Note: Es ist geplant Labels an einzelne Tortenstücke heften zu können. Aber das ist Zukunftsmusik.

7.7.1 Beispiele

Ein einfaches Chart mit fünf Segmenten:

```
<?xml version="1.0" encoding="UTF-8"?>
<presentation width="340" height="200">

  <widget name="ThePieChart" type="PieChart" location="10,10" shape="R 300 160">
    <property name="Values" type="List[Number]" value="2,4,3,5,7" />
<!--    <property name="Expansions" value="3" />-->
<!--    <template name="PieTemplates" type="Widget">-->
<!--      <widget name="pieSegment">-->
<!--        <property name="BackgroundColor" value="red" />-->
<!--      </widget>-->
<!--    </template>-->
  </widget>

</presentation>
```

7.7.2 Properties

Neben den Properties, die für alle Charts gelten (siehe Abschnitt *Gemeinsame Properties aller Charttypen*), hat das PieChart folgende eigene Properties:

Cutout (Number) Ein Kreischart kann in der mitte eine Aussparung haben. Cutout legt den Radius dieser Aussparung als Anteil am Radius des Kreises fest.

Default: `0`

Wertebereich: 0 bis 1

Values (List[Number]) Die Liste der Werte welche im PieChart dargestellt werden sollen. Die Summe aller Werte wird als 100% des Radius betrachtet. Somit beschreiben die einzelnen Werte den prozentualen Winkel. <– *Stimmt zwar, ist aber missverständlich formuliert. Oder hab ich da was falsch verstanden? / malte*

Beispiel:

```
<!-- PieChart mit 5 Segmenten mit jeweils 20% des Gesamtwinkels-->
<property name="Values" value="2,2,2,2,2"/>

<!-- PieChart mit 3 Segmenten mit jeweils 20%/40%/40% des Gesamtwinkels -->
<property name="Values" value="2,4,4"/>

<!-- PieChart mit 3 Segmenten mit jeweils 25% des Gesamtwinkels -->
<property name="Values" value="4,4,4,4"/>
```

Expansions (List[Number])

Einzelne Kreissegmente können mit dieser Liste nach aussen verschoben werden. Der Wert beschreibt den Versatz in Screeneinheiten (Pixel) nach außen; die Position in der Liste das jeweilige Kreissegment. Werden weniger Werte angegeben, als im Chart vorhanden sind, werden die Werte abwechselnd für die einzelnen Segmente genutzt:

```
<!-- alle Segmente mit einer Expansion von 5 -->
<property name="Values" value="3,4,2,6,7"/>
<property name="Expansions" value="5" />

<!-- Expansion immer abwechselnd mit 10 bzw. 5 -->
<property name="Values" value="3,4,2,6,7"/>
<property name="Expansions" value="10,5," />

<!-- Expansion für alle Segmente spezifisch gesetzt -->
<property name="Values" value="3,4,2,6,7"/>
<property name="Expansions" value="10,5,4,6" />
```

AnimationProgress (List[Number]) Eine Liste von Werten üblicherweise zwischen 0 und 1 die mit dem Winkel der einzelnen Kreissegmente multipliziert werden. Dadurch kann man individuelle Kreissegmente animieren. Sofern weniger Werte angegeben werden, als Segmente vorhanden sind, werden die überzähligen Segmente nicht animiert. *Wäre doch praktisch, wenn hier auch eine Automatismus wie bei Expansions funktionieren würde, also einen Wert angeben, für alle Segmente nutzen*

DefaultExpansion (Number) Mit dieser Eigenschaft lassen sich alle Segmente um den angegbenen Radius nach aussen rücken.

Default:`0`

Wo ist der Unterschied zwischen <property name="DefaultExpansion" value="10" /> und <property name="Expansions" value="10" /> Gibt es einen?/malte

ExpansionMode (Symbol) noch nicht definiert.

PieMode (Symbol)
- 'circle': Die Veränderung der Breite oder Höhe des Kreischarts ändert zwar den Radius der Segmente aber nicht den Aspekt. Es wird immer ein "rundes" Chart gezeichnet. Die Chartfläche wird in den zur Verfügung stehenden Raum eingepasst, unter Berücksichtigung der Property `Radius`.
- 'seats': Es werden nur 0 bis 180 Grad gezeichnet. (Halbkreischart). Das Chart wird in den zur Verfügung stehenden Raum eingepasst, unter Berücksichtigung der Property `Radius`.
- 'ellipse': Die Kreissegmente richten sowohl Breite als auch Höhe nach dem Kreischart aus. Das Chart füllt also den kompletten zur Verfügung stehenden Raum aus, unter Berücksichtigung der Property `Radius`.

Default: `ellipse`

StartAngle (Number) StartAngle definiert den Startwinkel des Charts in Grad. Dieser startet bei 0 Grad nach links und dreht im Normalfall gegen den Uhrzeigersinn. (Siehe Direction). Bei `PieMode='seats'` wird diese Eigenschaft ignoriert. (immer 0 Grad)

Default: `0`

EndAngle (Number) Definiert den Endwinkel des Charts in Grad. Bei `PieMode='seats'` wird diese Eigenschaft ignoriert. (immer 180 Grad)

Default: `360`

Direction (Symbol)
- 'clockwise' oder 'cw': Das Erhöhen von StartAngle oder EndAngle bewirkt eine Drehung im Uhrzeigersinn.
- 'counterclockwise' oder 'ccw': Das Erhöhen von StartAngle oder EndAngle bewirkt eine Drehung gegen den Uhrzeigersinn. (Standarteinstellung)

Default: `counterclockwise`

Groups (List[Object]) Über diese Eigenschaft lassen sich Gruppen von Kreissegmenten definieren die zwischen anderen Gruppen einen Spalt freilassen.

Radius (Number) Der Maximale Radius der Segmente. Dieser Radius kann größer werden wenn man Expansions einsetzt.

Section author: benjamin

7.8 LabelBarWidget

<property name="Orientation" value="vertical"/>

<bind property="Values" type="List[Date]" to="Controller" key="lists.headline"/>

<property name="Padding" value="0"/>

<property name="MaxLabels" value="50"/>

<template name="LabelTemplate" type="Text" shape="R 100 35"> <property name="BackgroundColor" value="0xffffff33"/> <property name="MultiLine" value="true"/> <bind property="Text" to="Controller.first" key="headline" /> <property name="Padding" value="10,4,4,18" /> <property name="TextFormat" value="@Bold13"/> <property name="Texture" value="images/bg_listitem.png"/>

</template>

Note: Chart-Animation:

siehe *Animation*

CHAPTER

EIGHT

MAONGO-MEDIA

Note: XXXXXXXXXXXXXXX Überarbeitung 25.7.2011 XXXXXXXXXXXXXXXXXXXX

8.1 MediaPlayerWidget

`maongo.core.toolkit.MediaPlayerWidget extends Widget`

Das MediaPlayerWidget dient der Wiedergabe von Audio und Video.

Ein einfaches MediaPlayerWidget:

```
<widget type="MediaPlayer" name="myVideoPlayer" shape="R 400 200">
        <property name="MediaType" value="video"/>
        <property name="Source" value="http://www.meinserver.de/video.mp4"/>
</widget>
```

8.1.1 Properties

AutoFallback (Boolean) Soll der Player automatisch nach Alternativen in den verfügbaren Quellen suchen, wenn keine Verbindung zustande kommt.

NYI

AutoFallbackTime (Number) Zeit in Sekunden nach der Alternativ-Quellen ausprobiert werden.

NYI

CurrentTime (Time, readonly) Liefert die momentane Zeitposition des wiedergegebenen Mediums.

LoopCount (Number) Anzahl der abzuspielenden Loops für das Medium.

MediaType (Symbol) Der Medientyp welcher wiedergegeben werden soll.

Default: `video`

Werte: `video`, `audio`

MetaData (Map, readonly) Liefert eine Map mit Metadaten zum aktuell wiedergegebenen Medium.

Hinweis:: Format und Inhalt des Objektes sind noch undefiniert

PlaybackControl (Symbol) Hiermit wird der momentane Abspielzustand gesetzt. Im Default-Zustand wird hierdurch ein Autoplay ermöglicht.

Default: `play`

Werte: `play`, `pause` , `stop`

```
<property name="PlaybackControl" value="play"/>
```

PosterFrame (Image) Ein Bild welches angezeigt wird, wenn das Medium noch nicht bzw. nicht angezeigt werden kann.

```
<property name="PosterFrame" value="[embed images/posterframe.png]"/>
```

Quality (String) Dient zur Ermittlung der Medienquelle aus einer evt. Liste mit mehreren Einträgen. Dabei muss der Eintrag in der Quelle mit dem Wert dieser Eigenschaft übereinstimmen, damit die Quelle als passend betrachtet wird. In der Defaulteinstellung `auto` wird jede Quelle als passend betrachtet.

Default: `auto`

Source (Media) Angabe welche Medienquelle wiedergegeben werden soll.

Momentan unterstützte Formate und Protokolle: Video: MP4 Dateien über HTTP-Protokoll und RTMP-Streams

Audio: MP3 Dateien über HTTP-Protokoll

Dabei kann entweder eine Quelle in der folgenden Syntax angegeben werden:

```
<property name="Source" type="Media" value="http://meinserver.de/v1.mp4"/>
```

Diese versuchen die Player auf den Plattformen wiederzugeben.

Um verschiedene Quellen für unterschiedliche Plattformen oder unterschiedliche Qualitäten einer Mediendatei anzugeben, ist es auch möglich eine Liste mit Quellen anzugeben. Diese wird von den Playern nach passenden Quellen durchsucht. Sollte dabei keine mit allen Kriterien (Plattform, Wiedergabeformate, Qualität) übereinstimmende Quelle gefunden werden, wird versucht eine für diese Plattform passende zu finden. Danach wird der erste Treffer aus der Liste abgespielt.

Syntax für mehrere Quellen:

```
<property name="Source" type="Media">
  <list>
    <map>
        <entry name="Protocol">RTMP</entry>
        <entry name="Type">video/mp4</entry>
        <entry name="Quality">high</entry>
        <entry name="URL">rtmp://rtmpserver/serverinstance</entry>
        <entry name="MediaPath" >mp4:meinevideoquelle.hi</entry>
        <entry name="UseFCSubscribe">true</entry>
        <entry name="Platform">Flash</entry>
    </map>

    <map>
        <entry name="Protocol">RTMP</entry>
        <entry name="Type">video/mp4</entry>
        <entry name="Quality">low</entry>
         <entry name="URL">rtmp://rtmpserver/serverinstance</entry>
        <entry name="MediaPath" >mp4:meinevideoquelle.lo</entry>
        <entry name="UseFCSubscribe">true</entry>
        <entry name="Platform">Flash</entry>
    </map>

    <map>
```

```
        <entry name="Protocol">HTTP</entry>
        <entry name="Type">video/mp4</entry>
        <entry name="Quality">high</entry>
        <entry name="URL">http://localhost:8888/vid/video2.mp4</entry>
    </map>

  </list>
</property>
```

Zur Beschreibung des Mediums können die folgenden Einträge in der Map übergeben werden:

URL (String) Die Url von der das Medium wiedergegeben werden soll. Es können HTTP, RTMP oder RTMPT-Quellen angegeben werden. Bei Angabe einer HTTP-Quelle ist die Angabe der Property `MediaPath` nicht notwendig. Für RTMP- bzw. RTMPT-Quellen ist hier die URL des Media-Servers anzugeben. Die Angabe des wiederzugebenden Mediums erfolgt in der Property `MediaPath`

Note: RTMP-Quellen werden derzeit nur in der Flash-Ausspielung unterstützt. Dabei wird eine Verbindung über RTMP und RTMPT aufgebaut und die schnellere Verbindung wird zur Wiedergabe genutzt.

MediaPath (String) Sofern RTMP/RTMPT-Quellen genutzt werden sollen, muss in dieser Property der für den Verbindungsaufbau notwendige Pfad zur tatsächlichen Medienquelle angegeben werden.

UseFCSubscribe (String) Soll für die Verbindung mit einem Media-Server über RTMP/RTMPT ein Subscribe Aufruf geschickt werden? Dies ist bei bestimmten Media-Servern notwendig um Medien abspielen zu können.

Default: `true`

Values: `true`, `false`

UsePolicyFile (String) Nur unter Flash bei der Wiedergabe von HTTP-Videos relevant. Sofern die Property auf `true` gesetzt ist, wird für das Video nach einem Cross-Domain-Policy File gesucht um zu prüfen, ob der Zugriff auf das Video erlaubt ist.

Default: `true`

Values: `true`, `false`

Protocol (String) Diese Eigenschaft legt fest, welche Art von Verbindung aufgebaut wird. Dies ist nur auf Plattformen relevant, welche neben HTTP auch RTMP/T/S unterstützen.

Values: `HTTP`, `RTMP`, `RTMPT`, `RTMPS`

Type (String) Art der Medienquelle um den Playern auf den Plattformen zu ermöglichen passende Formate zu finden.

Values: `video/mp4`, `video/ogg`

Platform (String) Zielplattform für die diese Quelle gedacht ist.

Values: `HTML`, `Flash`, `Java`

TextureMode

Mit der Eigenschaft "TextureMode" beeinflussen Sie die Art und Weise, mit der ein Medium in ein Widget eingepasst wird:

```
<property name="TextureMode" type="Symbol" value="center"/>
```

Der Standardwert der Eigenschaft TextureMode ist `top-left`.

off Das Medium wird nicht angezeigt. Audio wird trotzdem abgespielt.

Weitere Werte siehe *TextureMode*.

TotalTime (Time, readonly) Liefert die Gesamtlänge des wiedergegebenen Mediums. Sofern die Gesamtlänge des wiedergegebenen Mediums nicht bekannt ist, liefert diese Eigenschaft `-1` zurück.

VideoSize (RectangleXP, readonly) Liefert die "echte" Größe des Videos zurück, unabhängig von evtl. Skalierung. Sofern die Größe des Videos nicht bekannt ist, liefert diese Eigenschaft `null`.

Volume (Number) Lautstärke des wiedergegebenen Mediums. Sofern die Tonwiedergabe eines Mediums "getoggelt" wird, ändert sich das Volume nicht.

Default: `1.0`

Werte: `0.0` bis `1.0`

8.1.2 Methoden

play() Spielt das Medium ab bzw. startet die Wiedergabe nach einer Pause wieder.

pause() Pausiert die Wiedergabe des Mediums.

rewind() Spult das Medium an die Anfangsposition zurück. Dies entspricht einem `seek(0)`.

Note: Derzeit in Java nicht implementiert.

seek(targetposition) Springt an die in `targetposition` angegebene Zeitposition und verhält sich danach genauso wie vorher. Ein laufendes Medium wird nach einem `seek(10)` also weiterhin abgespielt, ein pausiertes Medium wird an der neuen Position pausiert. Angabe in Sekunden.

Note: Derzeit in Java nicht implementiert.

stop() Stoppt die Wiedergabe des Mediums. Das Medium beginnt nach einem folgenden `play()` wieder am Anfang. Durch den Aufruf von `stop()` wird der Buffer geleert und evt. die Verbindung zu einem Server getrennt. Es kann also nach einem erneuten `play()` etwas dauern bis die Wiedergabe erneut beginnt. Wenn ein "Soft-Stop" ausgeführt werden soll, ist dieser mit `pause(); seek(0);` zu erreichen.

toggleMute() Wechselt zwischen abgeschaltetem und angeschaltetem Ton. Dabei wird das Volume des Mediums nicht verändert. Nach dem Wiederanschalten ist der Ton also genauso laut wie vor dem Abschalten.

8.1.3 Signale

Signale sind weitgehend identisch auf allen Plattformen implementiert. Es kann allerdings zu unterschiedlichen Zeitpunkte oder Reihenfolgen beim Aussenden der Signale kommen, da die einzelnen Plattformen mit Medien sehr unterschiedlich umgehen.

media-buffer Der Buffering-Vorgang für ein Medium hat begonnen.

Note: Derzeit in Java nicht implementiert.

media-error Beim Laden oder Wiedergeben eines Mediums ist ein Fehler aufgetreten

media-finish Die Wiedergabe eines Mediums ist beendet.

media-pause Die Wiedergabe des Mediums wurde pausiert. Bei erneuter Wiedergabe erfolgt die Wiedergabe ab der momentanen Abspielposition.

media-play Die Wiedergabe eines Mediums hat begonnen. Dieses Signal wird entweder bei Wiedergabe nach einem Stop, nach einer Pause oder bei erstmaliger Wiedergabe des Videos ausgesendet.

media-seek Eine Änderung der Abspielposition wurde angestossen. Sobald die neue Abspielposition erreicht ist, verhält sich das Video wie vorher, wird also pausiert oder abgespielt.

media-stop Die Wiedergabe des Mediums wurde gestoppt. Sofern das Medium wieder abgespielt wird, erfolgt die Wiedergabe vom Anfang an.

In der HTML-Ausspielung erhält man aufgrund von Plattformspezifika nach einem `media-stop` noch eine `media-seek` Signal.

media-status Dieses Signal wird bei jeder Statusänderung des Mediums ausgesendet und beinhaltet den neuen Status des Mediums.

Values: `buffer`, `error`, `finish`, `pause`, `play`, `seek`, `stop`

8.1.4 Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<presentation xmlns="http://www.example.org/mad" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
width="1024" height="768">

<widget type="MediaPlayer" name="myVideoPlayer" shape="R 400 200">

  <property name="MediaType" value="video"/>
  <property name="LoopCount" value="0"/>
  <property name="PlaybackControl" value="play"/>
  <property name="PosterFrame" value="[embed images/posterframe.png]"/>
  <property name="TextureMode" value="top-left"/>
  <property name="Volume" value="0.8"/>
  <property name="Quality" value="high"/>

  <property name="Source" type="Media">
      <list>
          <map>
              <entry name="Protocol">RTMP</entry>
              <entry name="Type">video/mp4</entry>
              <entry name="Quality">high</entry>
              <entry name="URL">rtmp://rtmpserver/serverinstance</entry>
              <entry name="MediaPath" >mp4:meinevideoquelle.hi</entry>
              <entry name="UseFCSubscribe">true</entry>
              <entry name="Platform">Flash</entry>
          </map>

          <map>
              <entry name="Protocol">RTMP</entry>
              <entry name="Type">video/mp4</entry>
              <entry name="Quality">low</entry>
              <entry name="URL">rtmp://rtmpserver/serverinstance</entry>
              <entry name="MediaPath" >mp4:meinevideoquelle.lo</entry>
              <entry name="UseFCSubscribe">true</entry>
              <entry name="Platform">Flash</entry>
          </map>

          <map>
              <entry name="Protocol">HTTP</entry>
              <entry name="Type">video/mp4</entry>
              <entry name="Quality">high</entry>
              <entry name="URL">http://localhost:8888/vid/video2.mp4</entry>
          </map>

      </list>
```

```
  </property>

  </widget>
</presentation>
```

CHAPTER

NINE

DATA UND ROUTEN

Section author: jo

Note: <set property="..."> ist verwirrend. müsste <set name="..."> o.ä. heißen! jo! (*stimmt*)

Es gibt keine Entsprechung für define in JavaScript. (*genau!*) Wäre wünschenswert. (*Warum?* => Weil ich sonst in XML definen und in JS weiter damit arbeiten muss. Ein Ort wäre schöner.) `var` deklariert nur lokale Variablen. jo

Lookup im JavaScript hat sich im Vgl. zu MaongoCore verändert! (this-Parameter)

CHAPTER

TEN

DEFINES, LOOKUPS, BINDINGS

Section author: jo

10.1 What is it all about?

Wie andere Programmierumgebungen bietet auch MAD die Möglichkeit, Variablen zu definieren und auf diese zuzugreifen. In MAD wird eine Variable mit `define` im Kontext des jeweiligen Widgets deklariert. Daneben dienen auch die MAD-Tags `map` und `list` und eine Reihe weiterer Tags dazu, komplexe Variablentypen auf Widgets zu definieren.

Im Kontext des Widgets, auf dem die entsprechende Variable deklariert wurde, steht diese direkt als Objektvariable zur Verfügung, auf die mit der Objektreferenz `this` zugegriffen werden kann:

```
<widget>
        <define name="MyVar" />
        <action>
                trace(this.myVar);
        </action>
</widget>
```

Optional kann die Objektreferenz `this` auch weggelassen werden.

Wie aber greift man auf Variablen zu, die an anderer Stelle deklariert wurden? Dafür gibt es übergreifend für Widgets, Properties und Variablen einen einheitlichen Weg in MAD: Das “Lookup”. Lookups dienen dazu, auf Objekte an beliebiger Stelle in der MAD-Anwendung zuzugreifen.

Der Namensraum eines Widgets setzt sich zusammen aus

- seinen Properties
- seinen Child-Widgets (innerhalb des Widgets definierte weitere Widgets)
- den auf dem Widget deklarierten Variablen

Sobald wir ein beliebiges Widget in der MAD-Anwendung mit `lookup` adressiert haben, können wir auf alle diese im Namensraum des Widgets gespeicherten Informationen auf dieselbe Art und Weise zugreifen. (Ein weiterer Effekt ist, dass die Benennung von Child-Widgets und Variablen nicht mit den Widget-Properties und untereinander kollidieren darf!)

Manchmal ist es zudem wünschenswert, eine Widget-Eigenschaft dauerhaft an die Eigenschaft eines anderen Widgets oder an den Wert einer Variablen zu koppeln. Dieser Vorgang nennt sich Binding; er kann im MAD-XML mit dem `bind`-Tag ausgedrückt werden.

10.2 Lookups im XML

Lookups dienen dazu, Objekte (Widgets und ihre Properties, selbstdefinierte Variablen) innerhalb einer Maongo-Presentation zu adressieren. Soll ein Widget dieselbe Hintergrundfarbe erhalten wie ein zweites, so kann das so formuliert werden:

```
<widget name="W1" />
<widget name="W2">
        <property name="BackgroundColor" lookup="W1.BackgroundColor" />
</widget>
```

Widgets in einem anderen Zweig der Presentation müssen von einem gemeinsamen Parent aus adressiert werden:

```
<presentation>
  <widget name="W1">
    <widget name="W2">
      <widget name="W3" />

      <widget name="X1">
        <widget name="X2">
          <property name="BackgroundColor" lookup="W3.BackgroundColor" />
        </widget>
      </widget>

    </widget>
  </widget>
</presentation>
```

Das Lookup findet W3 im NameSpace des gemeinsamen Parent-Widgets W2. Dieselbe Referenz lässt sich im Beispiel ausdrücken durch:

```
<property name="BackgroundColor" lookup="W3.BackgroundColor" />
   (gemeinsamer Parent: W2)
<property name="BackgroundColor" lookup="W2.W3.BackgroundColor" />
   (gemeinsamer Parent: W1)
<property name="BackgroundColor" lookup="W1.W2.W3.BackgroundColor" />
   (gemeinsamer Parent: presentation)
```

10.3 Lookups in Attributen

Eine alternative Schreibweise zum `lookup`-Attribut ist, wie gewohnt das `value`-Attribut zu nutzen, aber ein `@` voranzustellen:

```
<property name="BackgroundColor" value="@W3.BackgroundColor" />
```

Diese Schreibweise erlaubt Lookups auch an Stellen, an denen kein Lookup-Attribut möglich ist. Beispiel:

```
<tween from="@startValue" to="@endValue" />
```

funktioniert, wenn die beiden Variablen `startValue` und `endValue` über Lookup erreichbar sind. Diese Syntax ist für alle Attribute erlaubt.

10.4 Lookups in JavaScript

Um ein Lookup durch die Hierarchie der Engines und Widgets zu machen, verwendet man die Funktion `$()`. Ein einfaches Lookup wie `$(this,"myvar")` würde zunächst in dem durch `this` referenzierten Widget nach der Property `myvar` suchen, danach im Parent des Widget und so weiter bis zur Presentation. Kann dei Property nicht gefunden werdn, wird eine Exception geworfen. dies lässt sich jedoch vermeiden, indem man einen Default-Wert mit angibt, der zurückgegeben wird wenn die Suche erfolglos bleibt. Der Default-Wert wird als dritter Parameter angegeben: `$(this,"myvar","")`

Komplexe Lookups

Lookups können verkettet werden, indem man mehrere Suchschlüssel mit Punkten verbindet: `$(this,"foo.bar.boo")` Die Property `foo` wird wie oben beschrieben in der Hierarchie gesucht, danach wird auf dem gefundenen Object eine Property `bar` angefragt und auf dem Resultierenden Objekt wiederum die Property `boo`.

TO BE CONTINUED

In Actions muss die Funktion `lookup()` (oder die Kurzschreibweise `$()`) benutzt werden, um das automatische Lookup hinauf zu den Widget-Parents zu erhalten. Ein direktes Referenzieren von Widgets (`this.WidgetName` oder `WidgetName`) ist nur möglich, wenn diese direkte Children des Widgets mit einer solchen Action sind:

```
<presentation>
  <widget name="W1">
    <widget name="W2">
      <widget name="W3" />

      <widget name="X1">
        <widget type="Button" name="X2">
          <action trigger="button-clicked">
              BackgroundColor = $(this, "W3").BackgroundColor; // OK
              BackgroundColor = $(this, "W3.BackgroundColor"); // OK
              BackgroundColor = W3.BackgroundColor; // Fehler:
                                                    // Widget W3 not fond
              BackgroundColor = X3.BackgroundColor; // OK
              BackgroundColor = this.X3.BackgroundColor; // OK
          </action>
          <widget name="X3" />
        </widget>
      </widget>

    </widget>
  </widget>
</presentation>
```

Die beiden Schreibweisen

```
BackgroundColor = $(this, "W3").BackgroundColor; // OK
BackgroundColor = $(this, "W3.BackgroundColor"); // OK
```

haben dabei eine unterschiedliche Herangehensweise, erreichen im Beispiel aber dasselbe.

Eine Referenz auf ein Widget (statt eine Property) per Lookup abzufragen, macht beispielsweise dann Sinn, wenn mehrere Eigenschaften des Widgets abgefragt oder gesetzt werden sollen:

```
<action>
        var w = $(this, "W3");
        w.Shape = "R 100,100";
```

```
        w.Location = w.Location + $$("Point", "10,20");
</action>
```

Die beiden Schreibweisen

```
var w = $(this, "W3");
var w = this.lookup("W3");
```

sind gleichbedeutend.

10.5 Defines

Defines sind auf einem Widget deklarierte Variablen.

Im Gegensatz zu Widget-Properties, die durch das Maongo-Framework vordefiniert sind und nicht vom Autor einer Applikation geändert werden können, bieten Defines die Möglichkeit, eigene Variablen zu definieren.

```
<define name="myLabelSpacing" type="Number" value="10"/>
```

Defines, Properties und auch benannte Childobjekte teilen sich einen Namespace. Das macht es erforderlich, dass auf demselben Widget derselbe Name nicht zweimal benutzt wird.

Zugriff auf eine oben definierte Variable:

```
<widget type="LabelSpace">
        <property name="LabelSpacing" lookup="sLabelSpacing" />
</widget>
```

10.5.1 Maps

Mehrere Definitions können zu einer `map` zusammengefasst werden:

```
<map name="DesignSet1">
        <define name="HeadlineTextColor" type="Color" value="0x4e4e4e"/>
        <define name="LegendeTextColor" type="Color" value="0x4e4e4e"/>
        <define name="HeadlineColorMarkShape" type="Shape" value="R 0 0"/>
</map>
```

Auf die Elemente der Map wird per Punktsyntax zugegriffen:

```
<widget>
   <property name="ForegroundColor" lookup="DesignSet1.HeadlineTextColor"/>
</widget>
```

10.5.2 Lists

Syntax?

Zugriff im XML?

10.5.3 Weitere im MAD deklarierbare Datentypen

Die folgenden Datentypen werden - wie einfache Defines - auf einem Widget definiert und sind sodann über Lookup erreichbar:

- `data` siehe *Data und Routen*.
- `animation`, `tween`, `sequence`, `parallel` siehe *Animation*.
- *weitere?*

10.6 Binding

Binding ist die dauerhafte Bindung einer Widget-Property an ein Datenobjekt. Ändert sich das Objekt, wird diese Änderung durch Binding auch an den Abnehmer weitergegeben.

Der `bind`-Tag nennt die gebundene `property`, das Datenobjekt (`to`) und die Objektvariable (`key`), deren Wert übernommen werden soll. Außerdem kann ein Defaultwert (`default`) angegeben werden, der greift, wenn das Datenobjekt oder der Zugriff auf `key` den Wert `null` ergibt:

```
<bind property="Text" to="Obj" key="ObjKey" default="MeinDefault" />
```

Das `to`-Attribut wird als Lookup interpretiert und kann enthalten:

- Lookup-Pfade zu Widgets
- Lookup-Pfade zu `DataObserver`-Properties von Widgets (siehe auch *Data und Routen*)
- Lookup-Pfade zu Maps

Wird an ein Widget gebunden, so sind alle Widgets-Properties als `key` möglich.

Beispiel für andere Datentypen:

Binden an das dritte Listenelement in einer Liste `antworten` in einem DataObserver:

```
<bind property="AnswerText" to="quiz.DataObserver" key="antworten[2]"/>
```

Binden an den Key `antwort2` einer Map:

```
<bind property="AnswerText" to="quiz.DataObserver.antwortMap"
        key="antwort2"/>
<bind property="AnswerText" to="quiz.DataObserver"
        key="antwortmap.antwort2"/>
```

Binden an ein Tabellenfeld (vgl. auch Data.Table in *Data und Routen*):

```
<bind property="AnswerText" to="quiz.DataObserver" key="Table[1][0]"/>
```

Binding in Chart-Elementen (beispielsweise des BarCharts): Hier können Eigenschaften der (Label-, Bar-) Templates an den Controller des Charts gebunden werden.

Zugriff mit `arrangedObjects.current`. Beispiel:

```
    <widget type="BarChart" name="myChart">
    <bind property="ControllerContent" to="ObservableData" key="Table"/>
    <property name="ColumnNames" type="List" value="label,color,value"/>
    <property name="BarValues" type="List[Number]"
```

```
            value="arrangedObjects.lists.value"/>

    <template name="Bar">
        <bind property="BackgroundColor" to="myChart.Controller"
            key="arrangedObjects.current.color"/>
    </template>

    <template name="Label">
        <bind property="Text" to="myChart.Controller"
            key="arrangedObjects.current.label"/>
    </template>
</widget>
```

Binding wird im XML der MAD-Presentation definiert. Es gibt keine JavaScript-Entsprechung.

10.7 Zuweisen von Werten in Skripten

In JavaScript-Actions erfolgt die Zuweisung mit dem =-Operator.

XML-Actions können mit dem `set`-Tag Werte zuweisen:

```
<action>
        <set property="PLabels.Visible" value="true" />
        <set property="PLabels.Visible" lookup="Settings.isLabelVisible" />
</action>
```

CHAPTER

ELEVEN

LAYOUT

Mit der Property `Layout` wird die Art beeinflusst, wie ein Widget seine Children layoutet. Diese Property ist auf dem Widget implementiert und steht allen Widget-Typen zur Verfügung.

Das Setzen dieser Property auf einem Widget bewirkt, dass auf diesem Widget ein Layoutmanager installiert wird, der alle direkten Child-Widgets in spezifischer Art layoutet; dabei werden u.U. Position und Shape der Child-Widgets verändert.

Es gibt verschiedene Arten von Layouts in Maongo. Dies sind

- Fill-Layout (die enthaltenen Children werden so positioniert das sie das Widget ausfüllen)
- Flow-Layout (die enthaltenen Children werden zeilenweise angeordnet)
- Box-Layout (die enthaltenen Children werden neben- bzw. übereinander angeordnet)
- Border-Layout (die enthaltenen Children werden auf die angegebenen Layoutpositionen platziert)

Syntax:

```
<property name="Layout" value="fill" />
<property name="Layout" value="box" />
```

11.1 Allgemeines

Der verfügbare Raum für alle Layouts ist per Default der rechteckige Bereich der `Bounds` (das umschließende Rechteck des auf einem Widget gesetzten Shapes) abzüglich eines evtl. `Padding`.

Die Abstände, die mittels des Parameters `gap` für die einzelnen Layouts definiert werden, wirken stets nur zwischen den gefüllten Bereichen eines Layouts, nie nach außen.

Widgets, deren Property `Visible` auf `false` steht, werden nicht dargestellt und auch im Layout nicht beachtet.

"PreferredBounds"

Zu jedem Layouttyp ist angegeben, welche `PreferredBounds` dieses Layout zurück liefert, wenn es danach gefragt wird. Hintergrund: Neben der Größenangabe, die mit `Shape` auf einem Widget gesetzt wird, kann ein Widget auch selbst bestimmen, welchen Platz es gerne hätte. Dies sind die `PreferredBounds`. Widgets mit Layouts fragen ihre Children nach `PreferredBounds`, um ein optimales Layout zu erzielen. Dieser Mechanismus funktioniert über mehrere Ebenen im Widget-Tree.

`PreferredBounds` sind eine Widget-Property und können auch explizit gesetzt werden. Siehe *Position, Größe und Form*. Werden `PreferredBounds` vom MAD-Autor gesetzt, so hat dies Vorrang gegenüber dem Widget-eigenen Mechanismus, sie zu bestimmen.

Die meisten Widget-Typen bestimmen ihre `PreferredBounds` mit der Größenangabe in `Shape` (oder dem Default-Shape, falls `Shape` nicht gesetzt ist). Ausnahme: Das Text-Widget ist in der Lage, seine `PreferredBounds` anhand der Textmenge zu bestimmen.

"LayoutConstraints"

Im Borderlayout müssen die ChildWidgets eines Widgets mit Layout mit der Property `LayoutConstraints` versehen werden, die dem Layout die spezifischen Positionen der einzelnen Child-Widgets mitteilt.

Auch im Filllayout kann die Property `LayoutConstraints` angegeben werden, um für einzelne Children ein spezifisches Layoutverhalten zu erreichen. Bei den anderen Layouts wird eine evtl. vorhandene Property `LayoutConstraints` auf den Children ignoriert.

11.2 FillLayout

```
<property name="Layout" value="fill" />
<property name="Layout" value="type:fill" />
```

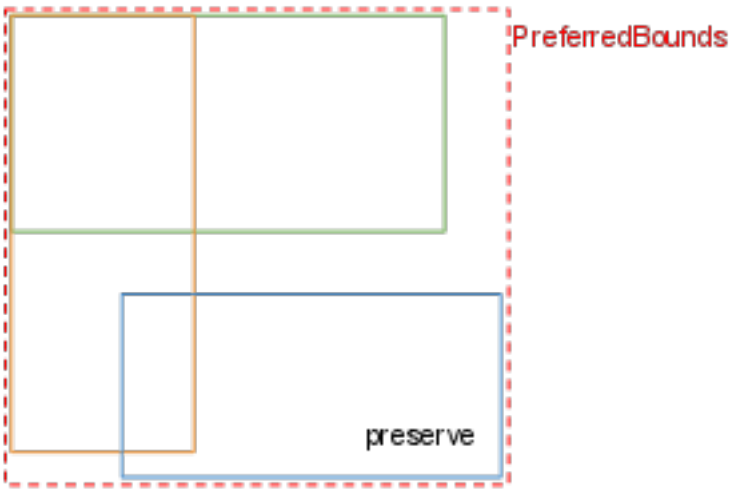
Enthält ein Widget, dessen Layout-Property auf `fill` gesetzt ist, weitere Widgets, so werden die Child-Widgets so positioniert, dass sie das Parent-Widget abzüglich eines eventuellen Paddings ausfüllen. Die Positions- und Shape-Angaben der Child-Widgets werden also ignoriert. Dieses Default-Verhalten kann durch LayoutConstraints auf den Child-Widgets verändert werden (s.u.)

Parameter

keine

PreferredBounds

Die PreferredBounds eines Widgets mit FillLayout werden berechnet, indem die größte Höhe und die größte Breite seiner Children g enutzt werden; gibt es Children mit `LayoutConstraints=preserve`, so wird auch die Location des betreffenden Childs mit eingerechnet. Children mit `LayoutConstraints=ignore` werden ignoriert



LayoutConstraints

Das Verhalten des FillLayouts kann durch die Angabe von LayoutConstraints auf den Childwidgets modifiziert werden. Bei Child-Widgets ohne LayoutConstraints wird der Default (`padding`) angenommen:

```
<property name="LayoutConstraints" value="padding" />
```

Das FillLayout nutzt je nach eingestellten LayoutConstraints unterschiedliche Modi, um die Größe eines Kindes anzupassen: Bei `resize` wird das Child-Shape vergrößert oder verkleinert (ein Kreis bleibt ein Kreis), bei `reshape` wird das Child-Shape vom Layout neu gesetzt, vom ursprünglichen Shape des Kindes bleibt nichts erhalten.

Mögliche Werte für LayoutConstraints sind:

- `padding`: das Shape des Child-Widgets wird so verändert (resize), dass es die Bounds des Parents-Shapes minus Padding ausfüllt (Default).
- `bounds`: das Shape des Child-Widgets wird so verändert (resize), dass es die Bounds das Parent-Shapes ausfüllt.
- `preserve`: keine Veränderung von Child-Shape und -Location.
- `ignore`: keine Veränderung von Child-Shape und -Location, widget wird bei der Berechnung der Preferred-Bounds ignoriert.
- `preferred`: keine Veränderung der Location, das Child-Shape wird auf seine PreferredBounds gesetzt (resize).

Beispiel

Ein Beispiel MAD mit verschiedenen LayoutConstraints: **MAD**.

11.3 FlowLayout

```
<property name="Layout" value="flow" />
<property name="Layout" value="type:flow; gap: 3" />
<property name="Layout" value="type:flow; gap: 3.5, 5.5" />
```

Enthält ein Widget, dessen Layout-Property auf `flow` gesetzt ist, weitere Widgets, so werden diese in horizontaler Richtung zei
angeordnet. Evtl. Positionsangaben im Child-Widget (x, y, Location) werden ignoriert.

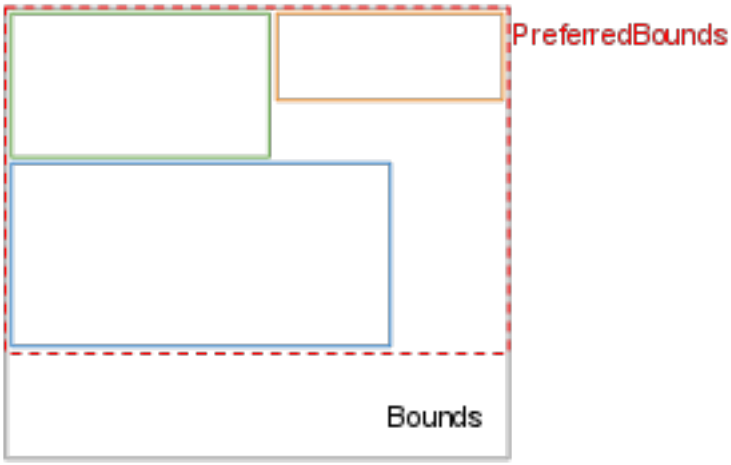
Das FlowLayout fragt seine Elemente nach ihren `PreferredBounds` und nutzt diese. Das hat zur Folge, dass beispielsweise Textwidgets den für die Darstellung des enthaltenen Textes nötigen Platz zurückmelden.

Parameter

```
gap: <Number>              Abstand zwischen Children: derselbe Wert für horizontal und vertikal
gap: <Number>, <Number>    Abstand zwischen Children: zwei Werte für horizontal und vertikal
gap: 0.0,0.0               (Default)
```

PreferredBounds

Die PreferredBounds eines Widgets mit FlowLayout werden zurückgeliefert als Breite der Original-Bounds und Höhe des für das Layout der Children inklusive der Gaps benötigten Bereichs.



LayoutConstraints

keine

Beispiel

Ein Beispiel **MAD**.

11.4 BoxLayout

```
<property name="Layout" value="box" />
<property name="Layout" value="type: box; gap: 5" />
<property name="Layout" value="type: box; direction: down; stackmode:equal; fillmode: preferred; gap
```

Enthält ein Widget, dessen Layout-Property auf `box` gesetzt ist, weitere Widgets, so werden diese nebeneinander oder übereinander angeordnet. Evtl. Positionsangaben im Child-Widget werden ignoriert. Für die Behandlung der Children-Shapes sind die Einstellungen in `stackmode` und `fillmode` ausschlaggebend.

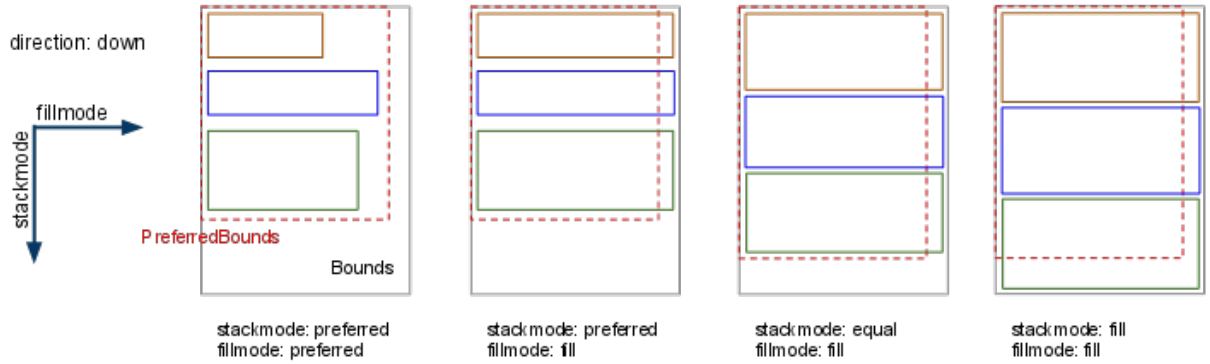
Parameter

```
direction: <up|down|right|left>     Richtung, in der das Layout gefüllt wird.
direction: down                     (Default) von oben nach unten

stackmode: <preferred|fill|equal>   Wie soll das Child-Widget gestapelt werden?
                                    Betrifft die Höhe bei direction=up/down und die Breite bei direct
stackmode: preferred                (Default) die entsprechende Dimension der PreferredBounds nutzen.
stackmode: fill                     Die Children so skalieren, dass sie den zur Verfügung stehenden P
                                    in dieser Dimension ausnutzen.
stackmode: equal                    Die Children so skalieren, dass sie in dieser Dimension so groß s
                                    größte Child-Widget.

fillmode: <preferred|fill>          Wie soll das Childwidget den zur Verfügung stehenden Platz ausfül
                                    Betrifft die Breite bei direction=up/down und die Höhe bei direct
fillmode: fill                      (Default) Die Children so skalieren, dass sie den zur Verfügung s
                                    in dieser Dimension ausnutzen.
fillmode: preferred                 Die entsprechende Dimension der PreferredBounds nutzen.
```

```
gap: <Number>                    Abstand zwischen Children
gap: 0.0                         (Default)
```

PreferredBounds

Die PreferredBounds eines Widgets mit BoxLayout werden zurückgeliefert als (bei direction=up/down) die Breite der Bounds/PreferredBounds des breitesten Elements und die Summe der Höhen der Bounds/PreferredBounds (zzgl. gaps) der Children, (bei direction=right/left) die Höhe der Bounds/PreferredBounds des höchsten Elements und die Summe der Breiten der Bounds/PreferredBounds (zzgl. gaps) der Children.



LayoutConstraints

keine

Beispiel

Ein Beispiel **MAD**.

11.5 BorderLayout

```
<property name="Layout" value="border" />
<property name="Layout" value="type:border" />
<property name="Layout" value="type:border; mode: wide; gap: 5" />
<property name="Layout" value="type:border; mode: wide; gap: 5,10,10,5" />
```

Ein Widget, dessen Layout-Property auf `border` gesetzt ist, kann bis zu fünf Child-Widgets auf die Layout-Positionen North, West, South, East und Center platzieren. Positions- und Shape-Angaben der Child-Widgets werden dabei u.U. modifiziert:

```
       |north |
- - - - - - - - - - -
west   |center| east
- - - - - - - - - - -
       |south |
```

Die fünf Layout-Positionen konkurrieren um den zur Verfügung stehenden Platz. Das Layout fragt stets die Preferred-Bounds der Children ab und versucht diesen Platzbedarf zu erfüllen. Ist `mode: wide` oder `mode:portrait`, so erhält die Layoutposition `center` den Platz, der nach Platzierung der anderen Laypoutpositionen übrig bleibt.

Ist das Attribut `mode:center`, so erhält die Layoutposition `center` den benötigten Platz, und die anderen Layout-positionen werden entsprechend angepasst.

Außerdem bestimmen die `mode`-Angaben darüber, wie die Bereiche in den "Ecken" des Layouts benutzt werden.

wide:

```
        north
- - - - - - - - - - -
west   |center| east
- - - - - - - - - - -
        south
```

portrait:

```
       |north |
       | - - -|
west   |center| east
       | - - -|
       |south |
```

Parameter

```
mode: <wide|portrait|center|centerwide|centerportrait>
mode: wide      (Default)

gap:  <Number>  (alle Richtungen)
gap:  <Number>, <Number>  (senkrechte Achsen, waagrechte Achsen)
gap:  <Number>, <Number>, <Number>, <Number> (obere, rechte, untere, linke Achse)
gap:  0         (Default)
```

PreferredBounds

Die `PreferredBounds` eines Widgets mit BorderLayout ist die Addition der Breiten und Höhen (zzgl. `gap`) aller `PreferredBounds` der Child-Widgets, nachdem sie entsprechend ihrer `LayoutConstraints` in das Layout platziert wurden.

LayoutConstraints

Innerhalb eines Widgets mit `Layout=border` müssen alle Children eine eindeutige Angabe in ihrer Property `LayoutConstraints` haben. Layoutpositionen können also nicht mehrfach belegt werden:

```
<property name="LayoutConstraints" value="south" />
<property name="LayoutConstraints" value="s" />
```

Für die Angabe der Himmelsrichtungen ist sowohl die Lang- als auch die Kurzschreibweise (`n, w, s, e, c`) möglich.

Beispiel

Ein Beispiel **MAD**.

CHAPTER

TWELVE

ANIMATION

Animationen sind zeitgesteuerte Veränderungen einer Presentation. Maongo besitzt ein Animationssystem, das es erlaubt, komplexe Animationsabläufe in XML zu definieren.

Die Animationen in Maongo sind zeitbasierend. Dies bedeutet, dass Animationen unabhängig von der Framerate, mit der die Presentation abgespielt wird, immer gleich lange dauern.

Note: Zeitlicher Ablauf einer Animation

Bei jedem Frameaufruf des Maongosystems werden in allen laufenden Animationen die aktuellen Animationswerte berechnet (zeitliche Position innerhalb der Animation, Werte der zu animierenden Eigenschaften etc.). Davon ausgehend werden auf den betroffenen Widgets die Werte gesetzt.

Soll eine Animation über einen Zeitraum von 4 Sekunden die Position eines Widgets von 0 auf 100 verändern, so bedeutet dies bei einer Framerate von 10 Bildern pro Sekunde eine Veränderung von 2,5 Pixeln pro Frame:

```
4 Sekunden à 10 Frames = 40 Frames; 100 Pixel / 40 = 2,5 Pixel
```

Sollte die Presentation mit 20 Frames pro Sekunde laufen, beträgt die Veränderung pro Frame nur 1,25 Pixel. Die Gesamtdauer der Animation bleibt unverändert.

12.1 Grundlagen

Eine einfache Animation lässt sich mit dem Element `<tween />` definieren, mit dem sich Widget-Eigenschaften über einen Zeitraum hin verändern lassen.

Beispiel:

```
<tween property="Location" type="Point" from="10,10" to="100,100" duration="1s"/>
```

Dieser Tween ändert die Widget-Eigenschaft `Location`; er bewegt das Widget im Zeitraum von einer Sekunde von Position `10,10` zu Position `100,100`. Die Animation wird durch das Setzen der Widget-Property `AnimationControl` auf den Wert `"play"` gestartet.

Beispiel:

```
<widget>
      <property name="Interactive" value="true"/>
  <tween property="Location" type="Point" from="10,10" to="100,100" duration="1s"/>
  <action trigger="pointer-down">
    this.AnimationControl = "play";
  </action>
</widget>
```

Animationselemente, die in einem Widget definiert sind, laufen parallel ab, wenn die Animation auf diese Art und Weise gestartet wird. Das Widget ist somit implizit ein Animationscontainer vom Typ `<parallel />` (s.u.).

Neben `<tween />` gibt es weitere "Animationsbausteine":

- `<switch />` schaltet die Widget-Eigenschaften direkt um;
- `<animation />` dient zum Aufruf einer Action in einer Animation bzw. zum Verweis auf an anderer Stelle definierte Animationen;
- `<wait />` dient zum Erzeugen einer Pause in einer Animation;
- `<iterator />` mittels eines Iterator lassen sich in einer Animation Listen mit Werten zur Verfügung stellen.

Daneben stellt Maongo zwei Elemente zur Verfügung, mit denen sich Animationen gruppieren lassen: `<sequence />` und `<parallel />`. Animationen innerhalb einer Sequence werden nacheinander, solche innerhalb einer Parallel-Animation gleichzeitig bzw. nebeneinander abgespielt.

Beispiel:

```
<sequence>
  <tween property="Location" type="Point" from="10,10" to="100,100" duration="1s"/>
  <tween property="Shape" type="Shape" from="R 10,10" to="R 100,100" duration="1s"/>
</sequence>
```

Die beiden Tweens laufen nacheinander ab, das Widget ändert zunächst seinen Ort, dann seine Form. Die Gesamtduer der Animation ist zwei Sekunden.

Beispiel:

```
<parallel>
  <tween property="Location" type="Point" from="10,10" to="100,100" duration="1s"/>
  <tween property="Shape" type="Shape" from="R 10,10" to="R 100,100" duration="1s"/>
</parallel>
```

Die beiden Tweens laufen parallel ab, Ort und Form werden gleichzeitig geändert. Die Gesamtdauer der Animation ist eine Sekunde.

Beispiel:

```
<sequence>
  <parallel>
    <tween property="Location" type="Point" from="10,10" to="100,100" duration="1s"/>
    <tween property="Shape" type="Shape" from="R 10,10" to="R 100,100" duration="1s"/>
  </parallel>
  <tween property="BackgroundColor" type="Color" from="red" to="blue" duration="1s"/>
</sequence>
```

Location und Shape werden gleichzeitig verändert, danach wird die Farbe von rot zu blau verändert. Das Beispiel zeigt auch, dass AnimationsContainer wie `sequence` und `parallel` ineinander geschachtelt werden können.

Ein weiterer AnimationsContainer ist `<iterator />`. Ein Iterator kommt immer dann zum Einsatz wenn innerhalb einer Animation eine Liste von Werten bzw. Widgets genutzt werden soll.

12.2 Gemeinsame Attribute

Die folgenden Attribute gelten für alle Animations-Elemente (tween, switch, animation, wait, iterator, sequence und parallel):

name (String) Bezeichnung des Elementes. Wird `name` benutzt, so müssen Animationen innerhalb desselben Containers eindeutig benannt sein. Der Name kann zum Aufruf der Animation in Skripten und für Lookups innerhalb der Anwendung genutzt werden.

target (String, @, {...}) Pfad auf ein Ziel-Widget, an dem die Animation ausgeführt werden soll.

Wird in der XML-Deklaration der Animation ein `target` gesetzt, so gilt dies für das Animation selbst und seine Kinder; setzen die Childanimationen selbst auch ein Target, so versucht die Animationsengine, im Namensraum des Parent-Targets das Child-Target aufzulösen.

Default: Wird target nicht gesetzt, so ist das Parent-Widget der Animation das Ziel-Widget der Animation.

Besonderheit: Ein per Skript übergebenes `target` wirkt nur, wenn auf der Animation noch kein `target` gesetzt ist.

duration (String, @, {...}) Netto-Dauer der Animation, ohne Pausen, Wiederholungen, etc. Wird die `duration` nicht angegeben, bestimmen die Parent- oder Child-Animationen die Dauer des einzelnen Elements. Als Werte können hier alle Angaben vom Datentyp `Time` interpretiert werden. Siehe dazu *Datentypen*.

Default: `1s` (eine Sekunde)

delay (String, @, {...}) Verzögerung bis zum Beginn der Animation. Eine Animation mit `delay="2s"` beginnt zwei Sekunden, nachdem sie gestartet wurde. Als Werte können hier alle Angaben vom Datentyp `Time` interpretiert werden. Siehe dazu *Datentypen*.

Default: `0`

count (Integer, @, {...}) Anzahl der Wiederholungen einer Animation.

Default: `1`

gap (String, @, {...}) Zeit zwischen zwei Wiederholungen. Als Werte können hier alle Angaben vom Datentyp `Time` interpretiert werden. Siehe dazu *Datentypen*.

Default: `0`

loopitems (List, @, {...}) Liste von Objekten, über die mit den Wiederholungen iteriert wird. Legt gleichzeitig die Anzahl der Wiederholungen auf die Länge der übergebenen Liste fest.

Ist `count` zusätzlich gesetzt, so gilt count, und die Liste wird entsprechend verkürzt genutzt, bzw. es wird erneut von vorne iteriert, bis `count` Wiederholungen erreicht sind.

extdur (String, @, {...}) Die (nach außen hin sichtbare) Dauer der Animation. Diese wird entweder aus der Dauer der einzelnen Bestandteile unter Berücksichtigung von Loops und Delays/Gaps berechnet, oder kann explizit gesetzt werden, um eine interne Verteilung der Gesamtzeit auf die beteiligten Animationen zu erreichen.

Als Werte können hier alle Angaben vom Datentyp `Time` genutzt werden. Siehe dazu *Datentypen*.

Im folgenden Beispiel ist jeder einzelne Tween 4 Sekunden lang (2 Tweens mit jeweils 4 Sekunden Länge):

```
<sequence extdur="8">
  <tween property="Location" type="Point" from="10,10" to="100,100"/>
  <tween property="Location" type="Point" from="100,100" to="10,10"/>
</sequence>
```

loop (String, @, {...}) Anzahl der Wiederholungen für diese Animation.

Default: `0`

Beispiel:

```
<sequence loop="2" >
  <tween property="Location" type="Point" from="10,10" to="100,100" duration="2s"/>
  <tween property="Location" type="Point" from="100,100" to="10,10" duration="2s"/>
</sequence>
```

ft: (Action, @, {...}) f(t, dur), Animationsfunktion. Eine Animationsfunktion muss für die Übergabewerte `t` (aktueller Zeitpunkt) und `dur` (Gesamtdauer der Animation), die als Parameter übergeben werden, einen Wert zwischen 0 und 1 zurückliefern.

Als mögliche Werte stehen vorgefertigte Funktionen zur Verfügung:

```
LinearNone, LinearIn, LinearOut, LinearInOut, QuadIn, QuadOut, QuadInOut,
CubicIn, CubicOut, CubicInOut, QuartIn, QuartOut, QuartInOut,
QuintIn, QuintOut, QuintInOut, BounceIn, BounceOut, BounceInOut,
BackIn, BackOut, BackInOut, CircIn, CircOut, CircInOut,
ElasticIn, ElasticOut, ElasticInOut, ExpoIn, ExpoOut, ExpoInOut,
SineIn, SineOut, SineInOut
```

Default: `LinearNone`

Es können auch Referenzen auf eigene Actions übergeben werden (Beispiel 2).

Beispiel 1:

```
<tween property="Location" type="Point" from="10,10" to="100,100"
  duration="5s" ft="QuadEaseIn"/>
```

Beispiel 2:

```
<tween property="Location" type="Point" from="10,10" to="100,100"
  duration="5s" ft="@MyEaseIn"/>
<action name="MyEaseIn" arguments="t,dur">
  return Math.pow(t/dur, 2);
</action>
```

Als dritte Möglichkeit können Funktionen direkt in die Animation geschrieben werden. Dabei stehen innerhalb des Scriptes die Variablen “t”(momentaner Zeitpunkt in der Animation) und “dur”(Dauer der Gesamtanimation) zur Verfügung.

Beispiel 3:

```
<tween property="Location" type="Point" from="10,10" to="100,100"
   duration="5s" ft="{return (t/dur)*t;}"/>
```

12.3 Gruppierung mittels `sequence`

Die in einem `<sequence />`-Container enthaltenen Animationen werden nacheinander abgespielt. Wird keine absolute Dauer angegeben, entspricht die Dauer der Summe der Bruttozeiten aller Children:

```
<sequence>
  <tween property="Location" type="Point" from="10,10" to="100,100" duration="1s"/>
  <tween property="Shape" type="Rectangle" from="10,10" to="100,100" duration="1s"/>
</sequence>
```

12.4 Gruppierung mittels `parallel`

Container, dessen Children gleichzeitig abgespielt werden. Die Dauer entspricht der Dauer der längsten Child-Animation. Hat ein Child keine Dauer, wird die Dauer des Containers genutzt (in diesem Fall 3 Sekunden, da die längste Child-Animation drei Sekunden lang ist):

```
<parallel>
  <tween property="Location" type="Point" from="10,10" to="100,100" duration="3s"/>
  <tween property="Shape" type="Rectangle" from="10,10" to="100,100" />
</parallel>
```

Attribute:

keine.

12.5 Animationselement `tween`

Veränderung einer Widget-Property über die Zeit:

```
<tween property="BackgroundColor" type="Color" from="#ff0000"
to="#00ffff" duration="3s"/>
```

Attribute:

property Name der property, die auf dem Widget (bzw. dem als `target` angegebenen Widget) verändert werden soll.

type (String) Datentyp für die `from`, `to`, `by`-Angaben.

Nicht jeder Datentyp kann sinnvoll getweent werden. Möglich sind beispielsweise: Integer, Number, Decimal, Point, Padding, Color, Shape. Es können nur Shapes getweent werden, die gleichartig sind (gleicher Typ / gleiche Anzahl Punkte etc.).

from (Object, @, {...}) Ausgangswert des Tweens.

to (Object, @, {...}) Endwert des Tweens.

12.6 Animationselement `switch`

Veränderung einer Widget-Property in Form eines einfachen Umschaltens. Besonders geeignet für den Datentyp `Boolean`.

```
<switch property="BackgroundColor" type="Color" from="#ff0000"
to="#00ffff" />
```

Attribute:

siehe Tween.

Ist beim Switch eine duration angegeben, so wird sie ignoriert. Ein Switch hat per Definition `duration=0`.

12.7 Animationselement `animation`

Dieser "Animationsbaustein" dient zur Integration von externen Elementen in eine Animation. Dies können Actions oder an anderer Stelle im Mad-Dokument notierte Animationen sein.

Attribute:

reference (Animation, @, {...}) Eine Animation, die an die Stelle des `animation`-Tags in den aktuellen Kontext eingefügt wird.

action (String, @, {...}) Eine Action. Als Werte können entweder ein Lookup auf eine im Mad definierte, benannte `<action />` oder ein Script übergeben werden. Nur wenn dieses Attribut genutzt wird, wirken sich die Attribute `mode` bzw. `duration` aus.

mode (Symbol) Dieses Attribut wirkt sich nur aus, wenn mittels `action` eine Action in die Animation integriert wird. Wird der `mode` auf "once" gesetzt, wird die Action nur einmalig aufgerufen. "onceifdefined" bewirkt, dass die action einmalig aufgerufen wird, aber nur wenn sie definiert ist. Eine nicht definierte Action wirft keinen Fehler. Wenn dieses Attribut weggelassen wird, erfolgt der Aufruf der Action auf jedem Frame innerhalb der gesetzten Dauer.

duration (Time, @, {...},INFINITE) Dieses Attribut wirkt sich nur aus, wenn mittels `action` eine Action in die Animation integriert wird. Dann bestimmt es die Zeit während der diese Action aufgerufen wird. Als Sonderfall gibt es den Wert "INFINITE". Dieser sorgt für ein endloses Laufen der Animation.

12.7.1 Aufruf von Actions

Wenn innerhalb einer Animation eine Action aufgerufen werden soll, kann dieses über das `animation`-Tag erfolgen.

Mittels des Attributes `mode` kann zwischen einem einmaligen Aufruf und einem wiederholten Aufruf unterschieden werden.

Im folgenden Beispiel wird die Action "FrameAction" für die Dauer von 50 Sekunden bei jedem Frame aufgerufen:

```
<animation action="@FrameAction" duration="50s" />
<action name="FrameAction" arguments="widgetref, val">
        // ...
</action>
```

Mittels des Attribut `mode` kann ein einmaliges Aufrufen der Action bewirkt werden. Wenn dieses Attribut weggelassen wird, erfolgt der Aufruf der Action auf jedem Frame innerhalb der gesetzten Dauer. Sofern eine Duration für ein `animation`-Tag angegeben ist, welches mittels `mode` auf einmalige Ausführunge geschaltet ist, wird die Duration als Wartezeit nach der Ausführung interpretiert.

Note: Bei einer einzelnen <animation> Anweisung ist die Angabe duration="0" notwendig, damit sie ausgeführt wird. Wenn das `animation`-Tag bspw. in eine `sequence` integriert ist, ist dieses nicht notwendig.

// geht

<animation action="@MyAction" mode="once" duration="0" />

// geht nicht

<animation action="@MyAction" mode="once" />

siehe Bug #281.

Einmaliger Aufruf einer Action in einer Sequence:

```
<sequence>
  <.../>
  <animation action="@MyAction" mode="once"/>
  <.../>
</sequence>
<action name="MyAction">
    // ...
</action>
```

Beispiel einer endlos laufenden Frameanimation:

```
<animation action="@MyAction" duration="INFINITE"/>
<action name="MyAction">
    // ...
</action>
```

Übergabewerte an die aufgerufene Action:

Die Deklaration der Action soll zwei Argumente vorsehen:

```
<action name="FrameAction" arguments="widgetref, val">...</action>
```

wobei in `widgetref` das aktuelle Target-Widget, in `val` der aktuelle Animationswert zwischen 0 und 1 übergeben wird. Die Action kann dann `val` nutzen, um beliebige Manipulationen am Target-Widget vorzunehmen. Für Inlinescripte "{...}" kann auf die Argumente unter diesen Namen zugegriffen werden.

12.7.2 Verweis auf andere Animationen

Erlaubt die Angabe einer Animation (Tween, Sequence, etc.), die an die Stelle des `animation`-Tags in den aktuellen Kontext eingefügt wird:

```
<widget name="widget1">
  <property name="Interactive" value="true"/>
  <property name="BackgroundColor" value="red" />
  <action trigger="pointer-down">
     this.AnimationControl = "play";
  </action>

  <sequence>
   <tween property="Shape" type="Rectangle" from="100,100" to="30,30"  duration="2s"/>
   // Verweis auf eine Animation welche in der Presentation definiert ist
   <animation reference="Presentation.HinHer"/>
  </sequence>
</widget>

// eingebundene Animation
<sequence name="HinHer">
   <tween property="Location" type="Point" from="0,0" to="100,100"  duration="2s"/>
   <tween property="Location" type="Point" from="100,100" to="0,0"  duration="2s"/>
</sequence>
```

12.8 Animationselement `wait`

Mittels des `wait` Containers kann eine Pause in laufende Animationen eingebaut werden:

```
<wait duration="2s" />
```

Im folgenden Beispiel wird zunächst die `Location` animiert, dann wird 5 Sekunden gewartet und anschließend wird das `Shape` animiert:

```
<sequence>
      <tween property="Location" type="Point" from="0,0" to="100,100"  duration="3s"/>
      <wait duration="5s" />
      <tween property="Shape" type="Rectangle" from="100,100" to="20,20"  duration="2s"/>
</sequence>
```

12.9 Animationselement `iterator`

Ein Iterator stellt innerhalb einer Animation eine Liste von Werten zur Verfügung die bei jedem Animationsdurchlauf auf das nächste Element weitergeschaltet wird.

Attribute:

items (List, @, {...}) Eine Liste der im Iterator enthaltenen Werte.

Im folgenden Beispiel wird nach Klick auf `widget1` eine Animation gestartet, die sowohl `loopitems` als auch einen `<iterator />` nutzt.

Als "Loopitems" der Animation sind die Childwidgets von `widget2` angegeben. Damit wird festgelegt, dass die Animation zweimal loopt (da `widget2` zwei Children hat); der Zugriff auf die einzelnen loopitems erfolgt über das `target="item"`.

Innerhalb der Animation ist ein `<iterator />` definiert, der über die Children von `widget3` iteriert. Der Zugriff auf diese Referenzen erfolgt über den Iterator-Namen (hier `myIterator`).

Das folgende Beispiel führt nacheinander folgende Aktionen aus:: widget2.child1 > Shapeanimation widget3.child1 > Visible widget2.child2 > Shapeanimation widget3.child2 > Visible

`widget3.child3` wird nicht verändert.

Beispiel:

```
<widget name="widget1">
  <property name="Interactive" value="true"/>
  <property name="Shape" value="R 20 20"/>
  <action trigger="pointer-down">
    this.AnimationControl = "play";
  </action>

  <sequence loopitems="@widget2.Children">
    <iterator name="myIterator" items="@widget3.Children"/>
    <tween target="item" property="Shape" type="Shape" from="R 20,20"
      to="R 50,50" duration="3s" />
    <switch target="myIterator" type="Object" property="Visible"
      from="true" to="false" />
  </sequence>
</widget>
```

```
<widget name="widget2">
  <property name="BackgroundColor" value="transparent"/>
  <property name="Shape" value="R 300 300"/>

  <widget name="child1"  x="100" y="50">
    <property name="Shape" value="R 20 20"/>
    <property name="BackgroundColor">red</property>
  </widget>

  <widget name="child2" x="200" y="50">
    <property name="Shape" value="R 20 20"/>
    <property name="BackgroundColor">green</property>
  </widget>
</widget>


<widget name="widget3" y="70">
  <property name="BackgroundColor" value="transparent"/>
  <property name="Shape" value="R 300 300"/>

  <widget name="child1"  x="100" y="50">
    <property name="Shape" value="R 20 20"/>
    <property name="BackgroundColor">red</property>
  </widget>

  <widget name="child2" x="200" y="50">
    <property name="Shape" value="R 20 20"/>
    <property name="BackgroundColor">green</property>
  </widget>

  <widget name="child3" x="300" y="50">
    <property name="Shape" value="R 20 20"/>
    <property name="BackgroundColor">green</property>
  </widget>
</widget>
```

12.10 Animationsspezifische Widget-Properties

AnimationControl (Symbol) Erlaubt die Steuerung der auf dem Widget definierten Animation(en).

Werte: `"play"`, `"pause"`, `"stop"`

12.11 JavaScript-Syntax

Die Animation des aktuellen Widgets abspielen:

```
this.AnimationControl = "play";
```

Eine benannte Animation abspielen:

```
Animationreference.play(target);
```

Im folgenden Beispiel wird das aktuelle Widget (`this`) von der Animation `this.moveIt` bewegt:

```
<widget name="widget1">
  <property name="Interactive" value="true"/>
  <property name="BackgroundColor" value="red" />

  <action trigger="pointer-down">
    this.moveIt.play(this);
  </action>

  <sequence name="moveIt">
    <tween property="Location" type="Point" from="0,0" to="100,100"  duration="3s"/>
  </sequence>
</widget>
```

Um eine Animation auf einem anderen Widget auszuführen, wird dieses Widget als `target` im `play`-Aufruf der Animation übergeben:

```
<widget name="widget1">
  <property name="Interactive" value="true"/>
  <property name="BackgroundColor" value="red" />
  <action trigger="pointer-down">
    // Lookup auf das Zielwidget
    var wgt = $(this,"widget2");
    // Aufruf der Animation mit der in 'wgt' abgelegten Referenz auf "widget2"
    this.scaleIt.play(wgt);
  </action>

  <sequence name="moveIt">
    <tween property="Location" type="Point" from="0,0" to="100,100"  duration="3s"/>
  </sequence>
  <sequence name="scaleIt">
    <tween property="Shape" type="Rectangle" from="100,100" to="20,20"  duration="2s"/>
  </sequence>
</widget>


<!-- Ziel-Widget -->
<widget name="widget2" y="100">
  <property name="Interactive" value="true"/>
  <property name="BackgroundColor" value="green" />
</widget>
```

12.12 Zeitverhalten von Animationen

Die Gesamtdauer einer Animation setzt sich wie folgt zusammen:

```
(delay + (duration * loopcount) + (gap * (loopcount-1)))
```

Die Dauer von Animationen kann von "außen" (Parent-Animation) oder von innen (Child-Animation) bestimmt werden.

Ist auf einer "äußeren" Animation eine `duration` angegeben, welche zu kurz ist um die enthaltenen Animationen abzuspielen, wird die Animation bis zum Ende der "äußeren" `duration` abgespielt. Die evt. noch nicht abgespielten "inneren" Animationen werden verworfen. Sofern die "äußere" `duration` länger ist, als die enthaltenen Animationen, läuft die Animation einfach weiter, ohne dass Auswirkungen sichtbar werden.

Beispiele zur Illustration des Zeitverhalten:

```
// Animationsdauer:
// sequence undefiniert => parallel undefiniert => längster tween b: 2s
// => parallel 2s + tween c 1s => sequence ist 3 Sekunden lang
// Tween a ist bereits nach 1s zu Ende.
<sequence>
  <parallel>
    <tween name="a" property="Location" type="Point" from="10,10" to="100,100"
      duration="1s"/>
    <tween name="b" property="Shape" type="Rectangle" from="10,10" to="100,100"
      duration="2s"/>
  </parallel>
  <tween name="c" property="BackgroundColor" type="Color" from="red" to="blue"
    duration="1s"/>
</sequence>

// Animationsdauer:
// sequence 5s - tween 1s => parallel 4s => tweens a,b sind 4 Sekunden lang
<sequence duration="5s">
  <parallel>
    <tween name="a" property="Location" type="Point" from="10,10" to="100,100" />
    <tween name="b" property="Shape" type="Rectangle" from="10,10" to="100,100" />
  </parallel>
  <tween name="c" property="BackgroundColor" type="Color" from="red" to="blue"
    duration="1s"/>
</sequence>

// Animationsdauer:
// sequence undefiniert => parallel undefiniert => a,b undefiniert
// => c undefiniert => oberstes undefiniertes Element: Sequence
// => sequence 1s (Default) => parallel 0.5s ==> a,b 0.5s
// => c 0.5s
<sequence>
  <parallel>
    <tween name="a" property="Location" type="Point" from="10,10" to="100,100" />
    <tween name="b" property="Shape" type="Rectangle" from="10,10" to="100,100" />
  </parallel>
  <tween name="c" property="BackgroundColor" type="Color" from="red" to="blue" />
</sequence>

// Animationsdauer:
// sequence undefiniert => parallel undefiniert => a,b undefiniert
// => c 1s => oberstes undefiniertes Element: parallel
// => parallel 1s (Default) ==> a,b 1s
// => sequence 2s
<sequence>
  <parallel>
    <tween name="a" property="Location" type="Point" from="10,10" to="100,100" />
    <tween name="b" property="Shape" type="Rectangle" from="10,10" to="100,100" />
  </parallel>
  <tween name="c" property="BackgroundColor" type="Color" from="red" to="blue"
  duration="1s"/>
</sequence>


// Animationsdauer: 12s, dabei sind die enthaltenen Animationen nach 8s fertig
<sequence duration="12s">
```

```
  <tween property="Location" type="Point" from="0,0" to="100,100"  duration="4s"/>
  <tween property="Location" type="Point" from="100,100" to="0,0"  duration="4s"/>
</sequence>

// Animationsdauer: 4s, der zweite Tween wird nicht mehr ausgeführt.
<sequence duration="4s">
  <tween property="Location" type="Point" from="0,0" to="100,100"  duration="4s"/>
  <tween property="Location" type="Point" from="100,100" to="0,0"  duration="4s"/>
</sequence>
```

12.13 Beispiele

Ein einfaches Beispiel mit `delay`:

```
// der zweite Tween wird mit einer Verzögerung von 3 Sekunden gestartet
<widget>
  <property name="Interactive" value="true"/>
  <property name="BackgroundColor" value="red" />
  <action trigger="pointer-down">
     this.AnimationControl = "play";
  </action>

  <sequence>
   <tween property="Location" type="Point" from="0,0" to="100,100"  duration="2s"/>
   <tween property="Location" type="Point" from="100,100" to="0,0"  duration="2s"
     delay="3s"/>
  </sequence>
</widget>
```

Mittels `gap` wird eine definierte Pause zwischen den mit `loop` definierten Wiederholungen eingesetzt:

```
<widget>
  <property name="Interactive" value="true"/>
  <property name="BackgroundColor" value="red" />
  <action trigger="pointer-down">
     this.AnimationControl = "play";
  </action>

  <sequence loop="5" gap="2s">
   <tween property="Location" type="Point" from="0,0" to="100,100"  duration="2s"/>
   <tween property="Location" type="Point" from="100,100" to="0,0"  duration="2s"/>
  </sequence>

</widget>
```

Ein umfangreicheres Beispiel erstellt eine Bilder-Slideshow mit dem Ken-Burns-Effekt (http://en.wikipedia.org/wiki/Ken_Burns_effect):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<presentation width="600" height="600">
  <property name="Interactive" value="true"/>
  <action trigger="pointer-down">
    $(this,"showAnim").play(this);
  </action>
```

```
  <widget name="area" x="50" y="50" >
    <property name="Shape" value="R 400 300"/>
    <widget name="img1" x="-50" y="-50">
      <property name="Shape" value="R 800 600"/>
      <property name="Center" value="400,300"/>
      <property name="Texture" value="../images/image1.jpg"/>
    </widget>

    <widget name="img2" x="-100" y="-50">
      <property name="Shape" value="R 800 600"/>
      <property name="Center" value="400,300"/>
      <property name="Texture" value="../images/image2.jpg"/>
    </widget>

    <widget name="img3" x="-70" y="-70">
      <property name="Shape" value="R 800 600"/>
      <property name="Center" value="400,300"/>
      <property name="Texture" value="../images/image3.jpg"/>
    </widget>

    <widget name="img4" x="-200" y="-230">
      <property name="Shape" value="R 800 600"/>
      <property name="Center" value="400,300"/>
      <property name="Texture" value="../images/image4.jpg"/>
    </widget>
  </widget>

  <sequence name="showAnim">
    <parallel duration="5s" name="anim1">
      <tween target="area.img4" property="Location" type="Point"
        from="-200,-230" to="-30,-70" />
      <tween target="area.img4" property="Scale" type="Number"
        from="1" to="1.5" />
      <tween target="area.img4" property="Alpha" type="Number"
        from="1" to="0" />
    </parallel>
    <parallel duration="5s" name="anim2">
      <tween target="area.img3" property="Location" type="Point"
        from="-70,-70" to="-300,-140" />
      <tween target="area.img3" property="Scale" type="Number"
        from="1" to="1.3" />
      <tween target="area.img3" property="Alpha" type="Number"
        from="1" to="0" />
    </parallel>
    <parallel duration="5s" name="anim3">
      <tween target="area.img2" property="Location" type="Point"
        from="-100,-50" to="-170,-100" />
      <tween target="area.img2" property="Scale" type="Number"
        from="1" to="1.4" />
      <tween target="area.img2" property="Alpha" type="Number"
        from="1" to="0" />
    </parallel>
  </sequence>
</presentation>
```

Das komplette MAD-Dokument zum **Download**.

12.14 Besonderheiten bei Chartanimationen

Chartanimationen nutzen die hier beschriebenen Animationsfeatures und Sprachkonstrukte. Allerdings werden einige zusätzliche Verfahren benutzt, um sehr granular auf die Einzelelemente von Charts (Balken, Linien, Labels, etc.) zuzugreifen.

Das folgende Beispiel ist in einem PieChart definiert:

```
1 <sequence name="s">
2       <sequence loopitems="@AnimationElements" extdur="6" delay="0.5">
3               <iterator name="it" items="@Labels.ValueLabels.Children"/>
4               <iterator name="it2" items="@Labels.DiffLabels.Children"/>
5               <tween target="item" property="Progress"
6                 reldur="@Values[*][0]" type="Number" from="0" to="1"/>
7               <switch target="it" type="Object" property="TextFormat"
8                 from="@LabelBoldTrans" to="@LabelBoldBlue" duration="0"/>
9               <wait duration="0.1" />
10              <switch target="it2" type="Object" property="TextFormat"
11                from="@LabelBoldTrans" to="@LabelBoldBlue" duration="0"/>
12              <wait duration="0.4" />
13      </sequence>
14 </sequence>
```

Das Attribut `loopitems` wird in Zeile 2 mit dem Lookup auf `@AnimationElements` initialisiert. Damit stehen die "Hauptelemente" jedes Charts (hier: die Kreissegmente, in den anderen Charttypen die Balken, Säulen und Linien) in der Animation zur Verfügung, und die Anzahl der Wiederholungen der `sequence` ist auf die Anzahl der `AnimationElements` festgelegt.

In Zeile 5 wird mit `target="item"` auf das jeweilige Loopitem zugegriffen. In diesem Fall wird die (Winkel-)Größe des Kreissegments von unsichtbar auf den vollen darzustellenden Wert animiert (Tween der Property `Progress` von 0 bis 1). ** heißt die nicht AnimationProgress??? **

Interessant ist auch das Timing diese Animation: hier werden die darzustellenden Werte für das Timing benutzt, so dass ein kleiner Wert (= ein kleines Kreissegment) weniger Zeit erhält als ein großer: `reldur="@Values[*][0]"`. Dieser Zugriff auf Values ergibt eine Liste, z.B. `[32,23,12,10]`. Für eine Animation, bei der jedes AnimationElement unabhängig vom Wert dieselbe Animationszeit erhält, kann `reldur` eine Liste `[1,1,1,1,1,1,1,1]` zugewiesen werden, die mindestens so lang sein sollte wie die Maximalzahl der erwarteten Werte.

Mit jeweils einem `iterator` werden weitere Elemente der Chartdarstellung gesteuert (Zeile 3 und 4). In diesem Fall sind in einem Container Labels zwei Labelbars definiert, deren Children per `iterator` in der Animation zur Verfügung gestellt werden. Mit `target="it"` (dem Namen des ersten Iterators) kann sodann auf die Children von `@Labels.ValueLabels`, mit `target="it2"` entsprechend auf die Children von `@Labels.DiffLabels` zugegriffen werden.

Mit jedem Durchlauf der in Zeile 2 definierten Sequence wird ein darin definierter `iterator` ein Element weitergeschaltet. In Zeile 5 und 7 wird so den einzelnen Labels ein anderes Textformat zugewiesen.

Selbstverständlich können weitere Animationselemente (wie die `wait`-Anweisungen in Zeile 9 und 12) die Chartanmation vervollständigen. Die beiden Anweisungen bewirken, dass zwischem dem Umschlaten der beiden Labels eine kurze Pause ist (Zeile 9), und am Ende der Gesamtsequenz für einen Balken (Balken und zwei Labels verändert) eine etwas längere Pause (Zeile 12).

Werden die Labels - wie im BarChart - direkt im Chart definiert, so ist auch ein Zugriff über die Chartproperty `BarLabels` möglich:

```
<widget type="BarChart" name="TheBarChart">
        <sequence loopitems="@AnimationElements" extdur="6" delay="0.5">
                <tween target="item" property="Progress" reldur="@Values[*][0]"
```

```
                type="Number" from="0" to="1"/>
              <switch target="item.BarLabels[2]" type="Object" property="TextFormat"
                from="@LabelBold" to="@LabelBoldBlue" duration="0"/>
              <switch target="item.BarLabels[3]" type="Object" property="TextFormat"
                from="@LabelBold" to="@LabelBoldBlue" duration="0"/>
              <wait duration="0.4" />
        </sequence>
        <property name="BarLabels" value="[@HeadtextLabel,@TextLabel,@ValueLabel,@DiffLabel]"/>
        <!-- ... -->
</widget>
```

`target="item.BarLabels[2]"` schaltet hier das ValueLabel (index 2 der Liste BarLabels), `target="item.BarLabels[3]"` entsprechend das DiffLabel.

und das LineChart?

CHAPTER

THIRTEEN

ACTIONS

Section author: jo

Note: Signals

Signals und der <detect> tag sollten vor den Action abgehandelt werden.

Actions sind Scriptböcke in einer Maongo-Präsentation, die auf unterschiedliche Weise ausgeführt werden können: Durch direkten Aufruf, über einen "trigger", als Teil einer Animation oder indem ein Data direkt auf die Action gerouted wird.

Beispiel:

```
<action name="goRed">
        this.BackgroundColor = "red";
        trace("Hello, World!");
</action>
```

Skripte in einer Skriptingsprache sollen nicht vom XML-Parser behandelt werden und sollten daher in einem CDATA-Block stehen:

```
<action language="javascript">
<![CDATA[
        // ...
]]>
</action>
```

Die Scriptingsprache ist Javascript.

13.1 Trigger

Widgets senden Signale ("Signals") aus, mit denen Actions ausgeführt werden können indem der Signalname als Trigger gesetzt wird.

Beispiel Buttonwidget:

```
<widget type="Button">

        <property name="BackgroundColor" value="0x800000"/>

        <action trigger="button-pressed">
                this.BackgroundColor = "red";
        </action>
```

```
        <action trigger="button-released">
                this.BackgroundColor = "green";
        </action>

</widget>
```

Trigger, die auf System- oder User-Signals reagieren:

```
Data/Routen:
<action trigger="data">
<action trigger="data-changed">

ButtonWidget: Maus ist auf dem Widget gedrückt/losgelassen worden:
<action trigger="button-pressed">
<action trigger="button-released">
<action trigger="button-clicked">

Allgemein: Maus ist auf dem Widget gedrückt/losgelassen worden:
<action trigger="pointer-down">
<action trigger="pointer-up">
<action trigger="pointer-up-outside">

Animation/Frame-Events:
<action trigger="animation-play">
<action trigger="frame" > ???? immer oder nur bei laufender Animation?
<action trigger="animation-end">
```

aber auch selbstdefinierte Trigger sind möglich:

```
<widget>
        <action language="js">
                signal("mao-test");
        </action>
        <action trigger="mao-test">
                // tu was
        </action>
</widget>
```

13.2 Der `action`-Tag

```
- trigger
        - Eine Action kann auch einen Trigger haben,
        der auf einem anderen Widget registriert ist
- name
        - Direktaufruf als Methode des Widgets
- language
- target
```

Eine Action kann auch einen Trigger haben, der auf einem anderen Widget registriert ist:

```
<widget location="70,20" rect="160,30">
        <action trigger="btnOne:button-clicked">
                <set property="BackgroundColor" value="0xff0000"/>
        </action>
```

```
        <action trigger="btnOne:button-released">
                <set property="BackgroundColor" value="0x0000ff"/>
        </action>
</widget>
```

Dieses Widget sollte sich umfärben, wenn auf btnOne geklickt wird!

13.3 Der `set`-Tag

13.4 Actions = Methoden des Widgets

- Benannte Actions direkt aufrufen
- Was ist der Vorteil, signal und trigger zu verwenden, gegenüber Methodenaufrufen mit <action name=”Methodenname”> und Methodenname() in einer JS-Action?

13.5 Real-World-Beispiel

Hier fehlt noch text...

(inc_typ_kreis.mad):

- Eine Action in XML-Syntax, dann eine reihe von JavaScript-Actions.
- Die Actions “Setters” und “Scripts” werden durch entsprechendes Routing aufgerufen:

Eine Route geht direkt auf Setters: Dann ist kein trigger nötig, die Action wird ausgeführt, als (Default-)Argument enthält arg das geroutete Data. Die Zweite Route geht auf das Widget “Scripts”. Damit würde zunächst noch gar keine Action ausgeführt, es sei denn, sie hat einen entsprechenden Trigger (trigger=”data”). Die Action “SetTotalValue” hat keinen Trigger, wird also nicht automatisch getriggert. Wenn Du in Zeile 84 guckst, siehst Du, dass die Action wie eine Methode in JavaScript aufgerufen wird: Das funktioniert, denn alle Actions, die in einem (Widget-)Kontext definiert werden, können auch direkt als Methoden aus JavaScript aufgerufen werden.

13.6 Rausgeworfen

Note: Skripte im XML

Skripte in einer Skriptingsprache sollen nicht vom XML-Parser behandelt werden

und stehen daher in einem CDATA-Block:

```
<action language="javascript">
<![CDATA[
        // ...
]]>
</action>
```

In Actions kann der Autor einer Maongo/MP-Presentation in Abhängigkeit von User-Interaktionen, als Reaktion auf einlaufende Daten u.ä. Zustandsänderungen herbeiführen.

Einfache Actions können vollständig im XML definiert werden; ist mehr Flexibilität vonnöten, als die XML-Syntax erlaubt, so kann der Template-Author Actions auch in JavaScript verfassen.

Beispiel für eine einfache Maus-Action:

```
<widget name="btn" type="Button">
        <property name="BackgroundColor" value="0x800000"/>
        <action trigger="button-pressed" target="widget">
                <set property="BackgroundColor" value="0xff0000"/>
        </action>
        <action trigger="button-released" target="widget">
                <set property="BackgroundColor" value="0x800000"/>
        </action>
</widget>
```

Auf dem Button-Widget sind zwei Actions definiert, die auf die Trigger `button-pressed` und `button-released` eine Zustandsänderung - hier die Änderung der Hintergrundfarbe des Widgets - herbeiführen.

Eine einfache JavaScript-Action kann so aussehen:

```
<widget name="anim">
        <action trigger="frame" language="javascript" target="widget">
                this.Rotation = this.Rotation + 3;
        </action>
</widget>
```

Hier wird bei jedem `frame`-Event die Rotation-Eigenschaft des Widgets verändert.

Section author: jo

CHAPTER

FOURTEEN

SCHRIFTEN

Maongo stellt fünf verschiedene, virtuelle Schriftfamilien zur Verwendung in Presentations zur Verfügung:

`sans-serif` oder `sans` eine Schriftart ohne Serifen

`serif` eine Schriftart mit Serifen

`monospace` eine Schriftart mit nichtproportionalen Zeichen (“Typewriter”)

Beispiel für die in Maongo integrierten Schriften

```
<presentation width="230" height="230">

    <textformat name="TFSans">
      <property name="FontFamily" value="sans" />
    </textformat>

    <textformat name="TFSansSerif">
      <property name="FontFamily" value="sans-serif" />
    </textformat>

    <textformat name="TFSerif">
      <property name="FontFamily" value="serif" />
    </textformat>

    <textformat name="TFMonospace">
      <property name="FontFamily" value="monospace" />
    </textformat>

    <widget type="Text" x="10" y="10">
      <property name="Text" value="Hello World"/>
      <property name="TextFormat" value="@TFSans"/>
    </widget>

    <widget type="Text" x="10" y="30">
      <property name="Text" value="Hello World"/>
      <property name="TextFormat" value="@TFSansSerif"/>
    </widget>

    <widget type="Text" x="10" y="50">
      <property name="Text" value="Hello World"/>
      <property name="TextFormat" value="@TFSerif"/>
    </widget>

    <widget type="Text" x="10" y="110">
```

```
      <property name="Text" value="Hello World"/>
      <property name="TextFormat" value="@TFMonospace"/>
    </widget>

</presentation>
```

Weitere Schriftarten können mit Fontdeklarationen im `<presentation>`-Tag geladen werden.

Im folgenden Beispiel wird eine Schrift unter dem Namen `MyFont` definiert, welche für den Style `plain` und den Style `bold` jeweils unterschiedliche Schriftdateien einbindet. Diese werden in den Textformaten `TFPlain` und `TFBold` entsprechend genutzt.:

```
<presentation width="230" height="230">

  <typeface name="MyFont">
        <font style="plain" name="PlainFont" source="fonts/arial.ttf" />
        <font style="bold" name="BoldFont" source="fonts/bradley.ttf" />
      </typeface>

  <textformat name="TFPlain">
   <property name="FontFamily" value="MyFont" />
   <property name="FontStyle" value="plain" />
  </textformat>

  <textformat name="TFBold">
   <property name="FontFamily" value="MyFont" />
   <property name="FontStyle" value="bold" />
  </textformat>

  <widget type="Text" x="10" y="10">
       <property name="Text" value="Hello World"/>
       <property name="TextFormat" value="@TFPlain"/>
  </widget>

  <widget type="Text" x="10" y="30">
       <property name="Text" value="Hello World"/>
       <property name="TextFormat" value="@TFBold"/>
  </widget>

</presentation>
```

Weitere Verwendungsbeispiele siehe *TextWidget*.

Section author: generated by maongo.apps.MemberDocs

CHAPTER

FIFTEEN

PROPERTY-ÜBERSICHT (STAND: 19. DEZEMBER 2011)

15.1 Engine

Property	Access	Type	Default	Comment
AnimationControl	r/w	Symbol	stop	Symbols: ANIMATION
Data	r/w	Data	null	*Skip setValue test Skip ObserveProperty test*
DataObserver	r-o	Object	—	*Skip setValue test Skip ObserveProperty test*
Debug	r/w	Boolean	false	
Name	r/w	String	—	
Presentation	r-o	Widget	*???*	*Skip setValue test Skip ObserveProperty test*

15.2 Widget

Property	Access	Type	Default	Comment
BackgroundColor	r/w	Color	—	See: color inheritance.
BorderColor	r/w	Color	—	See: color inheritance.
BorderWidth	r/w	Number	0	
Center	r/w	Point	0,0	
Children	r-o	List[Widget]	*???*	*Skip setValue test Skip ObserveProperty test*
Clipping	r/w	Boolean	true	
ForegroundColor	r/w	Color	—	See: color inheritance.
Height	r-o	Number	100	*Skip setValue test Skip ObserveProperty test*
InnerShape	r-o	Shape	R 100 100	*Skip setValue test Skip ObserveProperty test*
Interactive	r/w	Boolean	false	
Layout	r/w	Layout	null	
LayoutConstraints	r/w	Object	null	
LayoutPrivateData	r/w	Object	null	*Change notification missing: LayoutPrivateData*
Location	r/w	Point	0,0	
Padding	r/w	Padding	null	
PreferredBounds	r/w	Rectangle	null	
Rotation	r/w	Number	0	
Scale	r/w	Number	1	
Shape	r/w	Shape	R 100 100	
Texture	r/w	Image	null	*Skip setValue test Skip ObserveProperty test*
TextureAlpha	r/w	Number	1	
TextureMode	r/w	Symbol	top-left	Symbols: TEXTURE_MODE
Visible	r/w	Boolean	true	
Width	r-o	Number	100	*Skip setValue test Skip ObserveProperty test*
X	r/w	Number	0	*Skip ObserveProperty test: derived property*
Y	r/w	Number	0	*Skip ObserveProperty test: derived property*
ZIndex	r/w	Integer	*???*	*Skip setValue test Skip ObserveProperty test*

15.3 Presentation

Property	Access	Type	Default	Comment
AboutTitle	r/w	String	*???*	*Skip setValue test Skip ObserveProperty test*
AboutURL	r/w	URL	*???*	*Skip setValue test Skip ObserveProperty test*
Base	r/w	URL	—	
FullscreenBounds	r/w	Rectangle	*???*	*Skip setValue test Skip ObserveProperty test*
IO	r/w	Object	*???*	*Skip setValue test Skip ObserveProperty test*
Title	r/w	String	No Title	

15.4 Button

Property	Access	Type	Default	Comment
ActionOnPress	r/w	Boolean	false	
DataMode	r/w	Symbol	route	Symbols: DATA_MODE
DataSource	r/w	URL	null	
Group	r/w	String	null	
Mode	r/w	Symbol	click	Symbols: BUTTON_MODE
State	r/w	Symbol	off	Symbols: BUTTON_STATES
Text	r/w	String	null	

15.5 Canvas

Property	Access	Type	Default	Comment
Effect	r/w	Symbol	none	Symbols: EFFECTS
EffectStrength	r/w	Number	0	*NYI'

15.6 Card

Property	Access	Type	Default	Comment
StaticCard	r/w	Boolean	true	

15.7 Comm

Property	Access	Type	Default	Comment
Autostart	r/w	Boolean	false	*Skip setValue test Skip ObserveProperty test*
Encoding	r/w	String	utf-8	*Skip setValue test Skip ObserveProperty test*
EngineState	r/w	Symbol	stop	*Skip setValue test Skip ObserveProperty test*
Interval	r/w	Time	null	*Skip setValue test Skip ObserveProperty test*
Password	r/w	String	null	*Skip setValue test Skip ObserveProperty test*
ProxyPassword	r/w	String	null	*Skip setValue test Skip ObserveProperty test*
ProxyURL	r/w	URL	null	*Skip setValue test Skip ObserveProperty test*
ProxyUser	r/w	String	null	*Skip setValue test Skip ObserveProperty test*
Timeout	r/w	Time	00:00:30	*Skip setValue test Skip ObserveProperty test*
URL	r/w	URL	null	*Skip setValue test Skip ObserveProperty test*
User	r/w	String	null	*Skip setValue test Skip ObserveProperty test*

15.8 Flash

Property	Access	Type	Default	Comment
FlashVars	r/w	Map	null	*Skip setValue test Skip ObserveProperty test*
URL	r/w	URL	null	*Skip setValue test Skip ObserveProperty test*

15.9 Http

Property	Access	Type	Default	Comment
Encoding	r/w	String	utf-8	*Skip setValue test Skip ObserveProperty test*
Interval	r/w	Time	null	*Skip setValue test Skip ObserveProperty test*
Password	r/w	String	null	*Skip setValue test Skip ObserveProperty test*
ProxyPassword	r/w	String	null	*Skip setValue test Skip ObserveProperty test*
ProxyURL	r/w	URL	null	*Skip setValue test Skip ObserveProperty test*
ProxyUser	r/w	String	null	*Skip setValue test Skip ObserveProperty test*
RequestMode	r/w	Symbol	parallel	*Skip setValue test Skip ObserveProperty test*
Timeout	r/w	Time	00:00:30	*Skip setValue test Skip ObserveProperty test*
URL	r/w	URL	null	*Skip setValue test Skip ObserveProperty test*
User	r/w	String	null	*Skip setValue test Skip ObserveProperty test*

15.10 Text

Property	Access	Type	Default	Comment
FixedWidth	r/w	Number	0	
Formatter	r/w	Symbol	plaintext	Symbols: FORMATTER_MODES
HorizontalScrollBar	r/w	Widget	*???*	*Skip setValue test Skip ObserveProperty test*
HorizontalScrollPosition	r/w	Number	0	
MaxLines	r/w	Integer	0	
MultiLine	r/w	Boolean	false	
Text	r/w	String		
TextAlign	r/w	Symbol	top-left	Symbols: ALIGN
TextFormat	r/w	TextFormat	null	*Skip setValue test Skip ObserveProperty test*
TextStyles	r/w	Map	null	*Skip setValue test Skip ObserveProperty test*
Value	r/w	Object	—	*Skip setValue test Skip ObserveProperty test*
VerticalScrollBar	r/w	Widget	*???*	*Skip setValue test Skip ObserveProperty test*
VerticalScrollPosition	r/w	Number	0	

15.11 ScrollWidget

Property	Access	Type	Default	Comment
HorizontalScrollBar	r/w	Widget	*???*	*Skip setValue test Skip ObserveProperty test*
PointerScrollStep	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorActive	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorColor	r/w	Color	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorMinExtent	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorPermanent	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorWidth	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollXPosition	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollYPosition	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
VerticalScrollBar	r/w	Widget	*???*	*Skip setValue test Skip ObserveProperty test*

15.12 ScrollBarWidget

Property	Access	Type	Default	Comment
DisplayScrollButtons	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
DisplayScrollGrip	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
DisplayScrollGripBackgro	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollButtonsColor	r/w	Color	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollGripBackgroundColo	r/w	Color	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollGripBackgroundThic	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollGripColor	r/w	Color	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollGripExtentFixed	r/w	boolean	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollGripMinExtent	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollGripThickness	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollOrientation	r/w	Symbol	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollPosition	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollStep	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*

15.13 Socket

Property	Access	Type	Default	Comment
PingInterval	r/w	Time	null	*Skip setValue test Skip ObserveProperty test*
ReconnectInterval	r/w	Time	null	*Skip setValue test Skip ObserveProperty test*

15.14 Stack

Property	Access	Type	Default	Comment
Index	r/w	Integer	-1	
Mode	r/w	Symbol	managed	Symbols: STACK_MODE
SelectedCardWidget	r-o	Widget	—	*Skip setValue test Skip ObserveProperty test*
Size	r-o	Integer	0	*Skip setValue test Skip ObserveProperty test*

15.15 Line

Property	Access	Type	Default	Comment
DrawBobbles	r/w	Boolean	false	
DrawLine	r/w	Boolean	true	
LineWidth	r/w	Number	1	
Miter	r/w	Symbol	cut	Symbols: MITER
Points	r/w	List[Point]	[]	*Skip ObserveProperty test: derived property*
Progress	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*

15.16 List

Property	Access	Type	Default	Comment
Gap	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
HorizontalScrollBar	r/w	Widget	*???*	*Skip setValue test Skip ObserveProperty test*
ItemExtent	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ItemTemplate	r/w	Template	null	*Skip setValue test Skip ObserveProperty test*
List	r/w	List[Object]	null	*Skip setValue test Skip ObserveProperty test*
Orientation	r/w	Symbol	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorActive	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorColor	r/w	Color	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorMinExtent	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorPermanent	r/w	Boolean	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollIndicatorWidth	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
ScrollPosition	r/w	Number	*???*	*Skip setValue test Skip ObserveProperty test*
Selection	r/w	List[Number]	*???*	*Skip setValue test Skip ObserveProperty test*
SelectionMode	r/w	Symbol	*???*	*Skip setValue test Skip ObserveProperty test*
SelectionValues	r-o	List[Object]	*???*	*Skip setValue test Skip ObserveProperty test*
VerticalScrollBar	r/w	Widget	*???*	*Skip setValue test Skip ObserveProperty test*

15.17 Peking

Property	Access	Type	Default	Comment
Lifetime	r/w	Time	00:00:32	*Skip setValue test Skip ObserveProperty test*
ProtocolVersion	r/w	Integer	1	*Skip setValue test Skip ObserveProperty test*

15.18 AbstractChart

Property	Access	Type	Default	Comment
AnimationElements	r-o	List[Object]	[]	*Skip setValue test Skip ObserveProperty test*
AnimationProgress	r/w	List[Number]	[]	*Change notification missing: AnimationProgress*
AutoScale	r/w	Symbol	detect	Symbols: AUTO_SCALE
Colors	r/w	List[Color]	null	
Controller	r/w	ChartController	—	*Skip setValue test Skip ObserveProperty test*
Groups	r/w	List[Object]	null	*Unexpected value upon change notification recieved: [a, b, c], expected: [[CachedValue [ObjectValue a] cache: a], [CachedValue [ObjectValue b] cache: b], [CachedValue [ObjectValue c] cache: c]]*
HorizontalRaster	r/w	List[KeyValue]		*Wrong value after setValue: null
Maximum	r/w	Number	100	
Minimum	r/w	Number	0	
RasterDebugColor	r/w	Color	null	
Values	r/w	Table[Number]	null	*Skip setValue test Skip ObserveProperty test*
VerticalRaster	r/w	List[KeyValue]		*Wrong value after setValue: null
ZeroLineMode	r/w	Integer	0	
ZeroLineOffset	r/w	Number	0	
ZeroLineWidth	r/w	Number	0	

15.19 ChartController

Property	Access	Type	Default	Comment
ColumnNames	r/w	List[String]	null	
Columns	r-o	Object	—	*Skip setValue test Skip ObserveProperty test*
DataTableView	r/w	String	null	
Table	r/w	Table	null	
ValueColumn	r/w	String	null	

15.20 BarChart

Property	Access	Type	Default	Comment
BarLabels	r/w	List[Template]	null	*Skip setValue test Skip ObserveProperty test*
BarMetrics	r/w	List[KeyValue]	null	*Wrong value after setValue: null*Unexpected value upon change notification recieved: maongo.charts.BarChartMetrics@335856a5, expected: [[CachedValue [ObjectValue [KeyValue a: 12.0]] cache: [KeyValue a: 12.0]], [CachedValue [ObjectValue [KeyValue b: 3.0]] cache: [KeyValue b: 3.0]]]*
BarTemplates	r/w	List[Template]	null	*Skip setValue test Skip ObserveProperty test*
ChartAlign	r/w	Symbol	center	Symbols: CHART_ALIGNES
ChartMetrics	r/w	List[KeyValue]	null	*Wrong value after setValue: null*Unexpected value upon change notification recieved: maongo.charts.BarChartMetrics@6ca084f8, expected: [[CachedValue [ObjectValue [KeyValue a: 12.0]] cache: [KeyValue a: 12.0]], [CachedValue [ObjectValue [KeyValue b: 3.0]] cache: [KeyValue b: 3.0]]]*
ElementLabels	r/w	List[Template]	null	*Skip setValue test Skip ObserveProperty test*
ElementMetrics	r/w	List[KeyValue]	null	*Wrong value after setValue: null*Unexpected value upon change notification recieved: maongo.charts.BarChartMetrics@1e79ed7f, expected: [[CachedValue [ObjectValue [KeyValue a: 12.0]] cache: [KeyValue a: 12.0]], [CachedValue [ObjectValue [KeyValue b: 3.0]] cache: [KeyValue b: 3.0]]]*
GroupLabels	r/w	List[Template]	null	*Skip setValue test Skip ObserveProperty test*
GroupMetrics	r/w	List[KeyValue]	null	*Wrong value after setValue: null*Unexpected value upon change notification recieved: maongo.charts.BarChartMetrics@3dbbd23f, expected: [[CachedValue [ObjectValue [KeyValue a: 12.0]] cache: [KeyValue a: 12.0]], [CachedValue [ObjectValue [KeyValue b: 3.0]] cache: [KeyValue b: 3.0]]]*
MaxNumberOfBars	r/w	Integer	12	
MinNumberOfBars	r/w	Integer	1	
Orientation	r/w	Symbol	up	Symbols: ORIENTATION

15.21 Grid

Property	Access	Type	Default	Comment
HorizontalLines	r/w	List[Number]	[]	
HorizontalStep	r/w	Number	0	
LineWidth	r/w	Number	1	
NumberHorizontalLines	r/w	Integer	0	
NumberVerticalLines	r/w	Integer	0	
VerticalLines	r/w	List[Number]	[]	
VerticalStep	r/w	Number	0	

15.22 LabelBar

Property	Access	Type	Default	Comment
Gap	r/w	Number	0	
LabelTemplates	r/w	List[Template]	null	*Skip setValue test Skip ObserveProperty test*
MaxLabels	r/w	Integer	-1	
Orientation	r/w	Symbol	horizontal	Symbols: LABELORIENTATION

15.23 LineChart

Property	Access	Type	Default	Comment
EffectiveSamplePositions	r-o	Table[Number]	null	*Skip setValue test Skip ObserveProperty test*
GridFromXAxisValues	r/w	Boolean	false	
GridFromYAxisValues	r/w	Boolean	false	
HitDistance	r/w	Number	0	
LineLabels	r/w	List[Template]	null	
LineTemplates	r/w	List[Template]	null	
NullValue	r/w	Number	-1	*Skip setValue test Skip ObserveProperty test*
SamplePositions	r/w	Table[Number]	null	
StepBase	r/w	Number	0	
XAxisAnchorTexts	r/w	List[Object]	null	
XAxisAnchors	r/w	List[Number]	null	
XAxisAutoScale	r/w	Symbol	detect	Symbols: AUTO_SCALE
XAxisController	r/w	ChartController	—	*Skip setValue test Skip ObserveProperty test*
XAxisLabels	r/w	List[Template]	null	
XAxisMaximum	r/w	Number	100	
XAxisMinimum	r/w	Number	0	
YAxisAnchorTexts	r/w	List[Object]	null	
YAxisAnchors	r/w	List[Number]	null	
YAxisController	r/w	ChartController	—	*Skip setValue test Skip ObserveProperty test*
YAxisLabels	r/w	List[Template]	null	

15.24 PieChart

Property	Access	Type	Default	Comment
Cutout	r/w	Number	0	
DefaultExpansion	r/w	Number	0	
Direction	r/w	Symbol	clockwise	Symbols: CLOCK_DIRECTION
EndAngle	r/w	Number	360	
ExpansionMode	r/w	Symbol	null	Symbols: EXPANSION_MODE
Expansions	r/w	List[Number]	[]	
PieMode	r/w	Symbol	ellipse	Symbols: PIE_MODE
PieTemplates	r/w	List[Template]	null	*Skip setValue test Skip ObserveProperty test*
Radius	r/w	Number	0	
StartAngle	r/w	Number	0	

15.25 MediaPlayer

Property	Access	Type	Default	Comment
AutoFallback	r/w	Boolean	true	*Change notification missing: AutoFallback*
AutoFallbackTime	r/w	Integer	10	*Change notification missing: AutoFallbackTime*
CurrentTime	r/w	Time	*???*	*Skip setValue test Skip ObserveProperty test*
LoopCount	r/w	Number	1	
MediaType	r/w	Symbol	video	Symbols: MEDIATYPE
MetaData	r/w	List[Object]	null	*Exception**Exception*
Mute	r/w	Boolean	false	*Exception**Exception*
PlaybackControl	r/w	Symbol	play	Symbols: PLAYBACKCONTROL
PosterFrame	r/w	Image	*???*	*Skip setValue test Skip ObserveProperty test*
Quality	r/w	String	auto	*Change notification missing: Quality*
Source	r/w	Object	*???*	*Skip setValue test Skip ObserveProperty test*
TotalTime	r/w	Time	*???*	*Skip setValue test Skip ObserveProperty test*
Volume	r/w	Number	1	

Section author: malte

CHAPTER

SIXTEEN

MESSAGES - MAONGO KOMMUNIZIERT MIT DER UMWELT

16.1 Kommunikation zwischen Maongo und der einbindenden Host-Umgebung

Häufig wird eine Maongo-Anwendung innerhalb einer anderen Anwendung ausgeführt. Dies kann eine HTML-Seite sein, eine Flex-Anwendung oder eine Java-Anwendung. Um die Kommunikation zwischen der Host-Umgebung und der Maongo-Anwendung zu ermöglichen, stellt Maongo definierte Schnittstellen zur Übermittlung von Nachrichten bereit. Diese `Messages` bestehen aus einem `MessageName` und einer beliebigen Anzahl von Parametern als Nutzlast. Dabei dient der `MessageName` zur Identifizierung der Nachricht und kann genutzt werden um die Nachricht in der Maongo-Anwendung bzw. der Host-Umgebung an die richtige Stelle weiterzugeben.

16.2 Setup der Host-Umgebung

Um diese Funktionalität nutzen zu können muss die Host-Umgebung entsprechende Funktionen (Javascript bzw. Java) implementieren. Die Funktionen werden bspw. von den Maongo-Compilern für HTML und Flash schon in die generierten HTML-Seiten integriert.

HTML-Ausspielung

Der folgende Code-Abschnitt erlaubt den Zugriff auf die Maongo-Anwendung über die Kurzfassung `document.mymaongoapp` indem eine Referenz auf die erzeugte Applikation an der entsprechenden Stelle im DOM-Tree abgelegt wird.

```
...
var app = new mymaongoapp(...);
// setup for messaging
document.mymaongoapp = app;
...
```

Hiermit wird eine Javascript-Funktion als Handler für `Messages` aus der Maongo-Anwendung registriert.

```
...
// setup message handler
document.mymaongoapp.onMessage = messageCallback;
...
```

in HTML eingebundener Flashfilm

Der folgende Javascript-Code-Abschnitt erlaubt den Zugriff auf die Maongo-Anwendung über die Kurzfassung `document.mymaongoapp` indem eine Referenz auf das eingebundene SWF-File an der entsprechenden Stelle im DOM-Tree abgelegt wird.

```
...
//allow external messaging to app
document.mymaongoapp = document.getElementById("mymaongoapp");
...
```

Da der Handler für `Messages` aus der Maongo-Anwendung erst auf dem Flashfilm registriert werden kann, wenn dieser geladen ist und das Javascript-API initialisiert hat, wird eine definierte Funktion in der HTML-Seite angelegt, welche den Namen des Handlers zurückliefert. Diese Funktion wird vom Flashfilm aufgerufen, sobald dieser vollständig geladen ist.

```
//called from maongo after loading
function sendMessageHandler(){
  document.mymaongoapp.registerMessageHandler("messageCallback");
}
```

Der Zugriff des Flashfilms auf die HTML-Seite mittels Javascript wird über den Parameter `allowScriptAccess` in der HTML-Einbindung gesteuert. Dieser ist per default mit dem Wert `sameDomain` gesetzt. Dies bedeutet das der Zugriff nur erlaubt ist, wenn Flashfilm und HTML-Dokument von der gleichen Domain ausgeliefert werden. Sollte dies nicht der Fall sein, so kann der Wert auf `always` gesetzt werden.

```
<param name="allowScriptAccess" value="sameDomain"/>
```

bzw.

```
<param name="allowScriptAccess" value="always"/>
```

Flex-Anwendung

Um Nachrichten aus der Maongo-Anwendung in der Flex-Umgebung zu erhalten, muss eine Funktion der Flex-Umgebung als Message-Handler registriert werden

```
maongoComponent.registerMessageCallback(messageCallback);
```

Java-Anwendung

TODO

16.3 Nachrichten an eine Maongo-Anwendung schicken

Nachrichten werden an die Maongo-Anwendung geschickt, indem auf der Anwendung die Funktion `message(messageName, parameterList)` aufgerufen wird. Dabei ist die Anzahl der Parameter beliebig. Diese müssen allerdings in einen Array verpackt werden.

HTML-Ausspielung

```
<a href="javascript:document.mymaongoapp.message('myMessageName', ['klaus', 56, 'hans'])">send Messag
```

in HTML eingebundener Flashfilm

```
<a href="javascript:document.mymaongoapp.message('myMessageName', ['klaus', 56, 'hans'])">send Messag
```

Flex-Anwendung Aus einer Flex-Anwendung werden Nachrichten an die Maongo-Anwendung geschickt, indem die MaongoFlexComponent genutzt wird.

```
maongoComponent.message("messageName",["hallo welt,"klaus",23]);
```

Java-Anwendung

TODO

16.4 Nachrichten aus einer Maongo-Anwendung empfangen

Wenn Nachrichten aus Maongo verschickt werden sollen, so kann dieses innerhalb des MAD-Dokument auf jedem Widget mittels einer Action geschehen.

```
<action trigger="button-pressed">
  this.externalMessage("externalMessageName", ["dieter", 56]);
</action>
```

Die Nachricht wird in der Host-Umgebung an den registrierten Message-Handler weitergegeben und kann dort entsprechend weiterverarbeitet werden.

HTML/in HTML eingebundener Flashfilm

```
// sample implementation for messageCallback
function messageCallback(messageName, args){
  alert("callback: " + messageName + " arguments: " + args);
}
```

Flex

```
public function messageCallback(...rest):void{
  trace("Message from Maongo: " + rest);
}
```

Java-Anwendung

TODO

16.5 Beispiel

MAD-Dokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<presentation width="800" height="800">

<widget type="Button" x="20" y="20">
      <property name="Shape" value="R 80 20"/>
      <property name="Text" value="reset"/>
      <action trigger="button-pressed">
              this.parent.Display.Text = "reset";
      </action>
</widget>

<widget type="Button" x="20" y="50">
```

```
      <property name="Shape" value="R 80 20"/>
      <property name="Text" value="ext. Mess."/>
      <action trigger="button-pressed">
              this.externalMessage("externalMessage", ["parameter1",56]);
      </action>
</widget>

<widget type="Text" name="Display" x="20" y="100">
      <property name="Text" value="nix"/>
      <property name="BorderWidth" value="1"/>
      <property name="Shape" value="R 400 20"/>
</widget>

<!-- dieser Trigger wird ausgeloest wenn von extern eine Nachricht mit dem Namen "messageName" ankomm
<action trigger="messageName" arguments="arg1, arg2">
      this.Display.Text = "Message received: " + args;
</action>

</presentation>
```

HTML Ausspielung

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:svg="http://www.w3.org/2000/svg">
<head>
  <title>No Title</title>
  <link rel="stylesheet" type="text/css" media="screen" href="./mp_messagesfonts.css" />
  <link rel="stylesheet" type="text/css" media="screen" href="./mp_reset.css" />
  <script src="./mymaongoapp-assets/mp_maongomp.js" type="text/javascript"></script>
  <script src="./mymaongoapp-assets/mp_descriptors.js" type="text/javascript"></script>
  <script src="./mymaongoapp.js" type="text/javascript"></script>
  <script type="text/javascript">

  // <![CDATA[


  window.onload = function () {
    var ellapsed = new Date().getTime() - startTime;
    var body = document.body;
    var projector = document.getElementById('projector');
    var rc = new maongo.render.dom.RenderContext(projector);
    var app = new mymaongoapp(rc);
    app.setDefaultRenderer("dom");
    app.addListener(
      {
      onPreloadComplete: function () {
        projector.appendChild( app.getDisplayObject() );
      }
      }
    );


  // setup for messaging
  document.mymaongoapp = app;
  document.mymaongoapp.onMessage = messageCallback;

  app.start();
  ellapsed = new Date().getTime() - startTime;
  }
```

```
  // sample implementation for messageCallback - replace with your own implementation
  function messageCallback(messageName, args){
    alert("callback: " + messageName + " arguments: " + args);
  }

  // ]]>
  </script>
</head>
<body>
  <a href="javascript:document.mymaongoapp.message('testMessage', ['para1', 56])">send message to Mac
  <div id="projector"></div>
</body>
</html>
```

Einbindung der Flash-Ausspielung

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>No Title</title>
 <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
 <script type="text/javascript">
 //[ ... ]

 // sample implementation for messageCallback - replace with your own implementation
  function messageCallback(messageName, args){
   alert("callback: " + messageName + " arguments: " + args);
 }

</script>


</head>
<body style="padding:0px;margin:0px;overflow:hidden;height=100%">

<a href="javascript:document.mymaongoapp.message('testMessage', ['para1', 56])">send message to Maong

<div style="height:800px; width:800px;">
 <object id="mymaongoapp"  type="application/x-shockwave-flash" data="messages.swf?" style="width:800
   <param name="movie" value="messages.swf?" />
   <param name="scale" value="noscale" />
   <param name="salign" value="LT" />
   <param name="FlashVars" value="" />
   <param name="bgcolor" value="#" />
   <param name="allowFullScreen" value="true"/>
   <param name="allowScriptAccess" value="always"/>
   <div>Dieser Inhalt ist nur mit dem Adobe-Flash-Plugin sichtbar.</div>
 </object>
</div>
<script>
  //allow external messaging to app
  document.mymaongoapp = document.getElementById("mymaongoapp");

  //called from maongoflash
 function sendMessageHandler(){
   document.mymaongoapp.registerMessageHandler("messageCallback");
 }
```

```
</script>
</body>
</html><
```

Section author: ?? / jo

CHAPTER

SEVENTEEN

INFRASTRUKTUR

17.1 Projektor

Der Projector (Klasse maongo.core.Projector) ist der Referenzplayer für Maongo-Presentations.

Aufruf-Parameter sind:

```
-f <url>
        MAD-Datei.

-data <url>
        Quelle für Initialisierungs- und Poll-Data

-updates <poll|peking>

-interval <interval>
        Interval in Sekunden für Data-Poll

-cache <verzeichnis>
        Verzeichnis für lokalen Cache der Initdatas

-cachemode <update|replace>

-cachestore safe|key
```

17.2 MaongoFlashCompiler - SWF aus MAD erstellen:

Der Flash-Output (=das SWF) kann mit dem Flash-Standalone-Player oder im Browser abgespielt werden. Falls Security-Fehler auftreten (local-with-filesystem), bitte über http (MAMP o.ä.) aufrufen.

*Wir brauchen noch folgende Compiler-Optionen: * warnings an/abschalten

Vom MAD zum SWF:

```
* MaongoMP, LibMP, Esperanto auschecken
* In ~/.profile nachtragen:
        # MaongoFlash
        export FLEX_HOME=/Users/jo/Documents/flex_sdks/3.4.0
        export ANT_OPTS="-Xms512m -Xmx512m"
        export MAVEN_OPTS=-Xmx1024m
        export JAVA_OPTS="-Xms512m -Xmx512m"
```

```
* cd (svn_workspace)/MaongoMP
* ant noversion
        (übersetzt LibMP/MaongoMP nach AS und kompiliert Java- und
        ActionScript-Versionen. SWC für MaongoFlash entsteht.)

* ./target/maongo-noversion/bin/mfc <options> ¬
  /Users/jo/Documents/svn_workspace2/GameShow/source/mad/gameshow.mad
        (kompiliert ein SWF aus gameshow.mad und schreibt es in dasselbe
        Verzeichnis)

<options> können sein (./mfc --h):
        -d <outputdirectory>
                # overrides targetfile directory or mad file directory
        -o <targetfile>
                # name.[as|swf|swc], if no -d option given will set
                outputdirectory to basename of targetfile
        -noswc
                # don't try to compile using a maongo.swc
        -src [path1:path2:...]
                # use colon spearated pathnames as source library paths
        -flex <flex-sdk-dir>
        -maongo <maongo-sdk-dir>
        -fontdir <path-to-font-dir>
        -externaldebugger
                # use an external debugger - MonsterDebugger at the moment
        -loglevel ALL||TRACE|DEBUG|INFO|WARN|ERROR|FATAL|OFF
                # level of log-messages traced in external debugger. Default: OFF
        -generatedebug
                # create a debug version Default derzeit: true
        -warnings
                # show warnings during compile Default derzeit: false
        -nonetwork
                # dont use network/switch security sandbox Default derzeit: false
        -useoptimize
                # use postlink optimization for SWF Default derzeit: false
        -html
                # generate an minimal HTML file with generated swf embedded
```

17.3 Externen Debugger mit SWF benutzen:

Wenn eine MAD Datei mit der Option "-externaldebugger" und einem entsprechenden "-loglevel" in ein SWF kompiliert wird, werden die Ausgaben des Logsystems auf den externen Monsterdebugger umgeleitet.

Der Monsterdebugger ist eine Adobe AIR Anwendung, welche hier: http://www.demonsterdebugger.com heruntergeladen werden kann. Sobald man den Debugger parallel zu einem laufenden Flashfilm öffnet, werden die Ausgaben aus dem Logsystem dort angezeigt.

Hinweis:

- es kann teilweise bei sehr vielen Ausgaben zu einer leichten Verzögerung zwischen SWF und Monsterdebugger kommen. Dies liegt dann nicht an der MAD/SWF Datei sondern an der Kommunikation zwischen SWF und Debugger.
- die Einfärbung der Ausgaben entspricht dem Loglevel im Maongo-Code.

Tipps:

- unter “Window” sowohl “Live Application” als auch “Monitor” abschalten. Dann fühlt sich das ganze etwas schneller an.
- der Suchfilter ist sehr hilfreich bei einer Vielzahl von Ausgaben.
- um immer die aktuellste Ausgabe zu sehen, bietet es sich an die Liste nach der Nummer bzw. der Zeit absteigend zu sortieren

Section author: malte

CHAPTER

EIGHTEEN

INTEGRATION VON MAONGO IN FLEX

Um aus MAD übersetzte Maongo Anwendung in eine Flex-Anwendung zu integrieren, steht die MaongoFlexComponent im Paket maongo.flex zur Verfügung. Diese ist eine in Actionscript geschriebene und in MXML verwendbare Flexcomponente, welche als Wrapper für die Maongo Anwendung dient.

18.1 Einbindung der MaongoFlexComponent in MXML

Die Einbindung erfolgt in MXML einfach mittels

```
<maongo:MaongoFlexComponent id="maongoComponent"/>
```

Hierfür ist Voraussetzung, das der Namespace **maongo** wie folgt in der jeweiligen Application bzw. Komponente definiert ist

```
xmlns:maongo="maongo.flex.*"
```

Neben den flexinternen Eigenschaften verfügt die Komponente über einige Schnittstellen zur Kommunikation mit bzw. Integration in die umgebende Flexanwendung. Diese sind:

- maongoSwf - Pfad zum SWf welches geladen werden soll und welches die Presentation beinhaltet
- setPresentation - zum setzen der abzuspielenden Maongo-Presentation
- getPresentation - zum Auslesen der derzeit abgespielten Maongo-Presentation
- loadPresentation - zum Laden einer Maongo-Presentation von einer URL als fertig kompiliertes SWF
- handleTransmission - ermöglicht das Nutzen der Komponente als Transmissionhandler *TBD*
- handleTransmissionError - ermöglicht das Nutzen der Komponente als Transmissionhandler *TBD*
- loadTransmission - lädt eine Transmission direkt auf der Presentation *TBD*
- handleData - zur Weitergabe von Data an die Maongo-Presentation *TBD*
- registerMessageCallback - erlaubt die Weitergabe von Messages aus Maongo an die umliegende Anwendung mittels registrierter Callbacks *TBD*
- unregisterMessageCallback - hebt eine Registrierung als Message-Callback auf *TBD*
- message - erlaubt es eine Message aus der umliegenden Anwendung an Maongo zu schicken *TBD*

18.2 Anzeige einer Maongo Presentation in der MaongoFlexComponent

Um eine in ein SWF kompilierte Maongo-Anwendung in der MaongoFlexComponent anzuzeigen, wird der Pfad zum Maongo-SWf an die Komponente übergeben

```
<maongo:MaongoFlexComponent id="maongoComponent" maongoSwf="maongo_flash.swf" />
```

Um eine MaongoPresentation im Source-Code in der MaongoFlexComponent anzuzeigen, wird eine Presentation an die Komponente übergeben

```
var presentation:Presentation = myMadFile.createPresentation();
maongoComponent.setPresentation(presentation);
```

Die übergebene Presentation wird innerhalb der Komponente initialisiert und abgespielt.

18.3 Laden von Transmissions auf der Komponente und innerhalb der Komponente - TODO

18.4 Weitergabe von Data an die MaongoPresentation - TODO

18.5 Messages - Kommunikation zwischen der MaongoFlexComponent und der umliegenden Flexanwendung

Sofern in einer Flexanwendung die MaongoFlexComponent eingebunden ist, können an diese Nachrichten übermittelt werden. Dies erfolgt mittels

```
maongoComponent.message("MessageName","hallo","welt");
```

In der MAD-Datei wird ein entsprechender Action-Trigger wie folgt definiert

```
<action trigger="MessageName">
        // Action code here
</action>
```

ab hier TBD

Für den Empfang von Messages aus Maongo muss zunächst ein MessageCallback auf der MaongoFlexComponent registriert werden.

Dies erfolgt mittels

```
maongoComponent.registerMessageCallback(this,messageCallback,"MessageName");
```

Über den Parameter MessageName können verschiedene Callbacks für unterschiedliche Messages registriert werden.

- *Search Page*