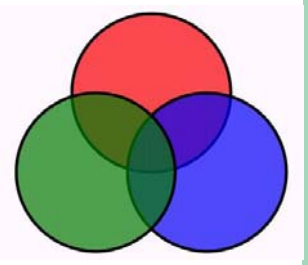
Vektorgrafik in Webseiten

Franz Fiala



Eine Reihe von Charts, die aus einer Datenbank generiert werden, sollen sowohl im Internet publiziert werden als auch mit CorelDraw für den Druck aufbereitet werden. Nun sind die Ansprüche für eine Internet-Darstellung und den Druck sehr verschieden und man könnte der Meinung sein, dass man dazu eben zwei verschiedene Programme benötigt. Doch im Sinne einer wirklich gut übereinstimmenden Darstellung von Webversion und Druckversion, sollte man von einer einheitlichen Quelle ausgehen. Gesucht war daher ein Format, das eine portierbare Vektor-Grafik erzeugt, die sowohl im Internet als auch im Druck zum Einsatz kommen kann.

Das Problem der Grafik-Darstellung im Internet ist die Herstellung und danach die Aktualisierung der Bilder. Wenn es sich zum Beispiel um Messwerte handelt, dann passen nach kurzer Zeit die Messwerte nicht mehr zu den Bildern. Jedes Chart, das in Excel generiert wurde, muss durch einen Screenshot in ein Bild verwandelt werden, um in einer Internet-Seite präsentiert werden zu können. Bei wenigen Grafiken ist diese Handarbeit kein Problem, es wird aber mühsam, wenn sich die Daten oft verändern und unmöglich, wenn es sich um Hunderte Grafiken handelt; ein Ausflug in die Welt der Grafik-Formate lohnt sich.

Man kann sich mit Formaten wie Adobe Flash behelfen und diese Formate dynamisch parametrieren. Dabei verlässt man aber die Welt des klassischen Html und verwendet proprietäre Formate, die nicht in jeder Umgebung verfügbar sind.

Dieser Beitrag zeigt, wie man in Web-Seiten portable Vektor-Grafik integrieren kann und wie man daraus dynamische Pixel-Grafiken erstellt.

SVG

Das gewünschte portierbare Format gibt es, es ist das Format *Scalable Vector Graphics* SVG und ist bei W3C normiert. Nach einigen Versuchen erweist sich das Format als sehr universell. Es ist in der Lage, sowohl Vektorgrafik, (auch Fonts) als auch Bitmaps darzustellen. Es wird von den Programmen Adobe Illustrator, Adobe InDesign, CorelDraw und vielen anderen als Export- und Import-Format akzeptiert. Browser interpretieren SVG mehr oder weniger gut bis gar nicht. Getestet wurden Firefox, Opera, Safari, Chrome und Internet-Explorer. Beim IE benötigt man zur Darstellung ein Plug-In. Die bestgeeigneten Browser für SVG-Darstellung ist Chrome oder Firefox.

Aber auch wenn man dieses Plug-In (von Adobe) am IE installiert, ist man über die mangelhafte Darstellung enttäuscht. Dazu kommt, dass damit praktisch alle Installationen des Internet-Explorers in Firmennetzen, bei denen die Installation von Plug-Ins verhindert wird, die allgemeine Verwendung des SVG-Format derzeit blockieren. Fast überflüssig zu erwähnen, dass auch eine Excel-Grafik nicht im SVG-Format exportiert werden kann. Kurz, die direkte Darstellung von SVG-Grafiken ist stark browserabhängig.

Und die Lösung?

Um daher eine wirklich portable Lösung zu erhalten, muss man die SVG-Datei zur Darstellung am Browser in ein JPG-Bild verwandeln. Diese Datei können dann natürlich alle Browser darstellen. Nicht mehr ganz so dynamisch, denn die Umwandlung in eine Grafik dauert auch einige Zeit. Mit dieser Umwandlung kann sich auch der bockigste Browser "brausen". Wir werden sehen, wie das im Einzelnen geschieht. Die SVG-Datei dient zur Publikation für den Druck. Da es sich um Vektor-Grafik handelt, kann jedes einzelne Element mit Adobe Illustrator oder CorelDraw nachbearbeiten (Farben, Positionen und Liniendicken nachjustieren uvam.).

Der folgende Beitrag zeigt die wesentlichen Elemente, die zur Herstellung portabler und dynamisch generierter Vektorgrafik notwendig sind. Alle Beispiele und alle Hilfsprogramme und Bibliotheken stehen bei der Webversion dieses Artikels zur Verfügung:

http://pcnews.at/pcn/1xx/11x/118/002400/_prg/

Demo http://fiala.member.pcc.ac/

Grundlegendes zum SVG-Format

Scalable Vector Graphics ist eine XML-Beschreibungssprache und daher – anders als HTML – nicht fehlertolerant. Sie erzeugt bei Verstößen gegen die Syntax-Regeln Fehlermeldungen. Die Schreibweise der Tags und Attribute entspricht den strengen Regeln von XML. Für den Anwender bedeutet das praktisch ein Aus für händische Kode-Generierung (außer bei einfachen Beispielen), weil der Interpreter fehlerbehaftete Kodes ignoriert und gröbere Verstöße mit Fehlermeldungen begleitet.

Beispiel für eine SVG-Datei
(Beispiel1.svg)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
  width="800" height="800">
 <g style="fill-opacity:0.7; stroke:black; width:0.1cm;">
  <circle cx="200" cy="100" r="100"
   style="fill:red;" transform="translate(0,50)" />
  <circle cx="200" cy="100" r="100"
   style="fill:blue;" transform="translate(70,150)" />
  <circle cx="200" cy="100" r="100"
   style="fill:green;" transform="translate(-70,150)"/>
 </g>
</svg>
```

Das `svg`-Tag definiert ein Bild mit den Abmessungen `width` und `height`. Das `g`-Tag bildet eine Gruppierung und weist dieser Gruppe gemeinsame Eigenschaften, hier Transparenz (`opacity`), Zeichenfarbe (`stroke`) und Zeichenbreite (`width`). Die dargestellten Grafikelemente sind drei überlappende Kreise mit den Mittelpunkten `cx` und `cy`, den Radien `r` und verschiedenen Füllfarben `fill`. Damit sich die gleich großen Kreise überlappen, werden sie durch das Attribut `translate` verschoben.

Die SVG-Datei `Beispiel1.svg` kann mit jedem Browser (außer mit dem Internet-Explorer) aufgerufen werden. Achtung: Der Unterschied zu einem Bild ist der, dass man diese Grafik nicht über das Kontext-Menü speichern kann, so, wie man das mit einem Bild gewohnt ist, denn es ist eben kein Bild. Einen Browser-Test, der auch die SVG-Kompatibilität in die Prüfung einbezieht, findet man hier: *http://acid3.acidtests.org/*.

Spezifikation

http://de.wikipedia.org/wiki/SVG

http://www.w3.org/Graphics/SVG/

http://www.w3.org/TR/SVG11/

Lehrgänge

http://luxor-xul.sourceforge.net/talk/jug-nov-2002/slides.html

http://www.w3.org/TR/2007/WD-SVGPrintPrimer12-20071221/

http://www.w3.org/TR/2007/WD-SVGPrint12-20071221/

http://svg.tutorial.aptico.de/start.php

http://www.w3schools.com/svg/

http://tutorials.jenkov.com/svg/

http://www.adobe.com/svg/basics/intro.html

http://www.datenverdrahten.de/svglbc/

http://www.kevlindev.com/tutorials/basics/index.htm

http://www.academictutorials.com/svg/

http://roitsystems.com/tutorial/

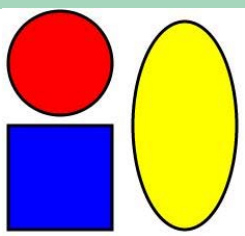
http://apike.ca/prog_svg.html

Die mit SVG darstellbaren Grafiken sind sehr vielfältig und die Mächtigkeit der Sprache kann man daran ablesen, dass man komplexe Grafiken aus Adobe Illustrator oder Corel-Draw in dieses Format exportieren kann und das Ergebnis ohne Qualitätsänderung im jeweils anderen Programm weiterbearbeitbar ist und auch im kompatiblen Browser darstellbar ist.

Der Nachteil ist, dass man jetzt zwar eine perfekte Grafik hat aber keinen Text, der sie erklären könnte. Man kann zwar Grafik-Text zur Beschriftung verwenden aber eine HTML-Seite wird es dadurch auch noch nicht,

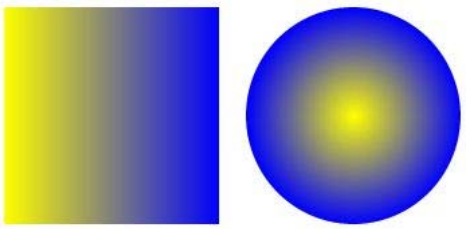
Grundformen: b10.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >
 <g style="stroke:black; stroke-width:3;">
  <circle cx="60" cy="60" r="50" style="fill:red" />
  <rect x="10" y="120" width="100" height="100" style="fill:blue" />
  <ellipse cx="180" cy="120" rx="50" ry="100" style="fill:yellow" />
 </g>
</svg>
```



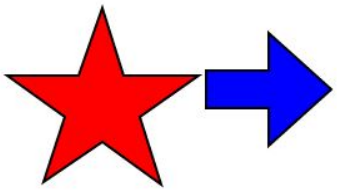
Gradient: b23.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >
 <linearGradient id="gradient1">
  <stop offset="0%"   style="stop-color: yellow" />
  <stop offset="100%" style="stop-color: blue" />
 </linearGradient>
 <radialGradient id="gradient2">
  <stop offset="0%"   style="stop-color: yellow" />
  <stop offset="100%" style="stop-color: blue" />
 </radialGradient>
 <rect x="10" y="10" width="160" height="160"style="fill:url(#gradient1)" />
 <circle cx="270" cy="90" r="80"style="fill:url(#gradient2)" />
</svg>
```



Polygon: b30.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >
 <polygon style="fill:red; stroke: black; stroke-width:2"
   points="97,20 114,74 174,74 127,107 144,161 97,127 50,159 67,107 21,74 78,74" />
 <polygon style="fill:blue; stroke: black; stroke-width:2"
   points="180,70 230,70 230,40 280,85 230,130 230,100 180,100"/>
</svg>
```



Text: b35.svg

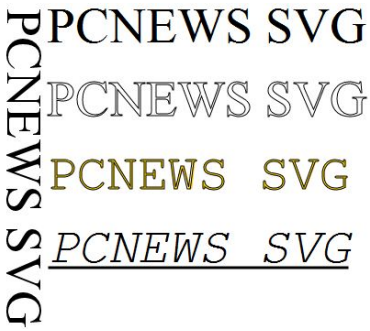
```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >
 <text x="60" y="60" style="font-size: 55;">PCNEWS SVG</text>

 <text x="60" y="140"
  style="stroke: black; fill: none; font-size: 55; font-family: 'Times New Roman'">
  PCNEWS SVG
 </text>

 <text x="60" y="220"
  style="stroke: black; fill: gold; font-size: 55; font-family: 'Courier New' ">
  PCNEWS SVG
 </text>

 <text x="60" y="300"
  style="font-size: 55; font-style: italic; font-family: 'Courier New';
      text-decoration: underline; ">
  PCNEWS SVG
 </text>

 <text x="30" y="10" style="font-size: 55; writing-mode:tb">PCNEWS SVG</text>
</svg>
```



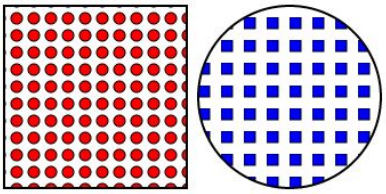
Filter: b36.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >
 <filter id="filtereffect1" height="150%">
  <feGaussianBlur in="SourceAlpha" stdDeviation="6" result="image1" /></filter>
 <filter id="filtereffect2" height="150%">
  <feGaussianBlur in="SourceAlpha" stdDeviation="6" result="image1" />
  <feOffset in="image1" result="image2" dx="5" dy="5" />
  <feComposite in="SourceGraphic" in2="image2" operator="over" /> </filter>
 <filter id="emboss" >
  <feGaussianBlur in="SourceAlpha" stdDeviation="2" result="blur"/>
  <feSpecularLighting in="blur" surfaceScale="-3" style="lighting-color:white"
  specularConstant="1" specularExponent="16" result="spec" kernelUnitLength="1" >
   <feDistantLight azimuth="45" elevation="45" /></feSpecularLighting>
  <feComposite in="spec" in2="SourceGraphic" operator="in" result="specOut"/>
 </filter>
 <g style="font-size:100; font-weight:bold">
  <text x="20" y="120" style="filter:url(#filtereffect1)">PCNEWS SVG</text>
  <text x="20" y="220" style="fill: orange; filter:url(#filtereffect2)">PCNEWS SVG
  </text>
  <text x="20" y="320" style="fill: orange; filter:url(#filtereffect2)">
   PCNEWS SVG
  </text>
  <text x="20" y="320" style="fill: orange; filter:url(#emboss)">
   PCNEWS SVG
  </text>
 </g></svg>
```

Füllmuster: b50.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >

 <pattern id="pattern1" x="0" y="0" width="15" height="15"
    patternUnits="userSpaceOnUse">
  <circle cx="6" cy="6" r="5" style="fill: red; stroke:black;" />
 </pattern>
 <pattern id="pattern2" x="0" y="0" width="20" height="20"
    patternUnits="userSpaceOnUse">
  <rect x="0" y="0" height="10" width="10" style="fill: blue; stroke:black;" />
 </pattern>
 <rect x="10" y="10" width="160" height="160"
   style="fill:url(#pattern1); stroke:black; stroke-width:2;" />
 <circle cx="260" cy="90" r="80"
   style="fill:url(#pattern2); stroke:black; stroke-width:2;" />
</svg>
```



Pfade: b60ori.svg

Originaldatei von http://de.wikipedia.org/wiki/Datei:Tux.svg

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" >
 <g style="fill:#c1c1bf;">
  <path d="M 329.336 727.552
        C 315.224 726.328 304.136 715.816 303.128 694.936
        C 306.368 639.496 309.608 582.112 271.232 545.104
        C 265.256 499.024 244.016 482.104 234.008 452.512
        L 218.24 441.208          L 237.104 411.688
        L 245.168 411.904         L 323.936 571.168
        L 340.424 651.448         L 329.336 727.552
        z" />
 </g>  ...
</svg>
```

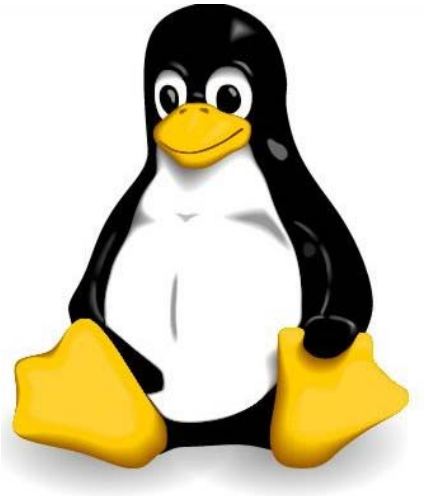


Diagramm1: b70.svg Leistung des SK-Rapid bei den letzten Bundesligaspielen. (Siehe http://rapid.iam.at/)

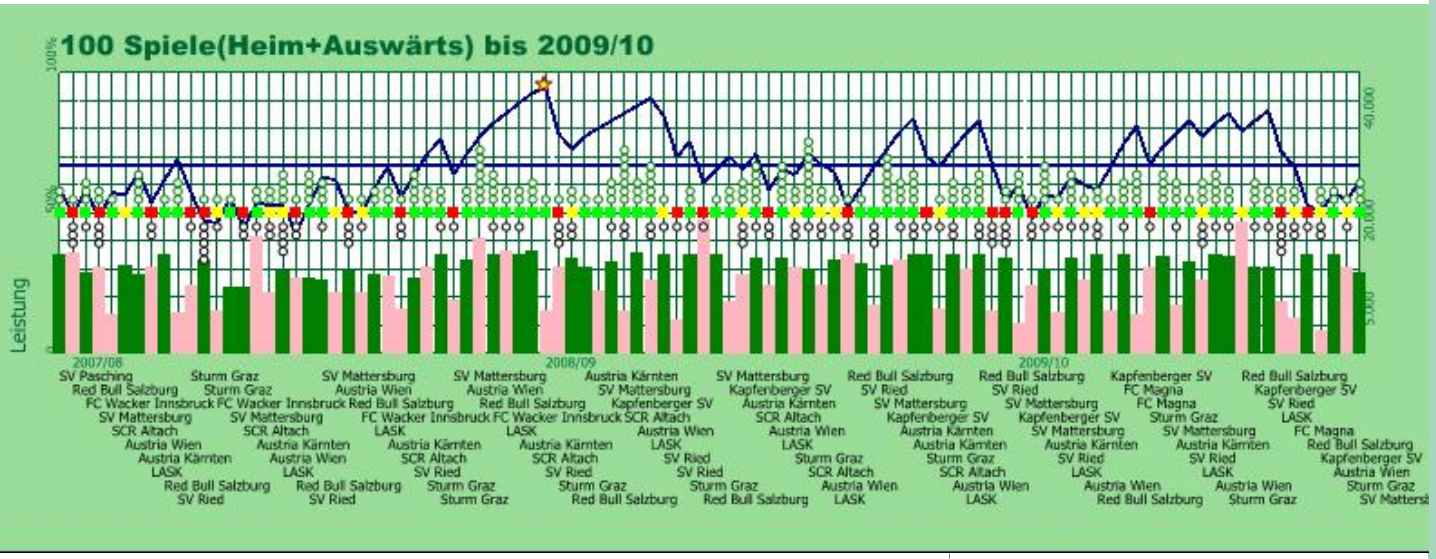
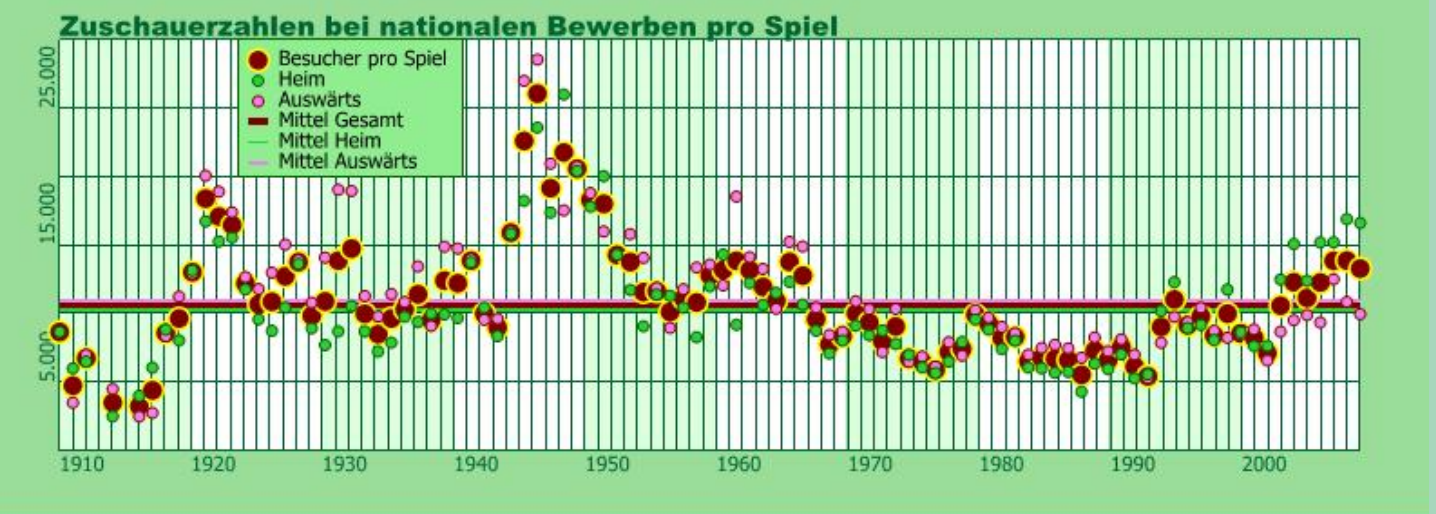


Diagramm2: b80.svg Zuschauerzahlen des SK Rapid (Siehe http://rapid.iam.at/statistics.aspx?id=graphics&id1=visitors)



denn man kann nicht Html-Elemente in das SVG-Format einfügen. Fügt man zum Beispiel `<p>Test</p>` ein, passiert gar nichts, denn das Element `p` ist in SVG nicht definiert und wird ignoriert. Macht man dann noch den Fehler, die einheitliche Kleinschreibung nicht zu beachten, wie in `<P>Test</p>`, erhält man eine Fehlermeldung.

Diese Beschränkung ist durch die DTD *(Document Type Definition)* in der zweiten Zeile des Kodes zu suchen:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

Es gibt aber auch Dokumentdefinitionen, die es erlauben, verschiedene Sprachen auf einer Seite zu mischen. Will man daher HTML und SVG-Grafik auf einer Seite kombinieren, muss man folgende DTD benutzen:

http://www.w3.org/TR/2002/WD-XHTMLplusMathMLplusSVG-20020809/

Jetzt erhält man eine XHTML-Datei als Rahmen, die es erlaubt SVG-Abschnitte, also Vektorgrafiken einzubetten:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
  "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
<html>
<body>
  <p>Dieser Text erscheint mit der Grafik auf einer Seite.</p>
  <svg xmlns="http://www.w3.org/2000/svg"
     width="800" height="800"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <g style="fill-opacity:0.7; stroke:black; width:0.1cm;">
      <circle cx="200" cy="100" r="100" style="fill:red;"
        transform="translate(0,50)" />
      <circle cx="200" cy="100" r="100" style="fill:blue;"
        transform="translate(70,150)" />
      <circle cx="200" cy="100" r="100" style="fill:green;"
        transform="translate(-70,150)"/>
    </g>
  </svg>
</body>
</html>
```

In diesem Kodebeispiel wird die vorher erzeugte Grafik in XHTML eingebettet. Der Grafik-Abschnitt wird durch das `svg`-Tag eingerahmt. Es geht aber auch umgekehrt, man kann auch HTML in eine SVG-Grafik einbetten; Details dazu im W3C-Dokument:

http://www.w3.org/TR/2002/WD-XHTMLplusMathMLplusSVG-20020809/#howto-svg

Warum eigentlich XHTML und nicht HTML?

XHTML1.0 ist eine strengere Version von HTML4.0. Browser sind fehlertolerant und ignorieren Fehler im HTML- (oder XHTML-)Kode. Es ist aber im Sinne der Präzision der Darstellung hilfreich, wenn eine strenge Überprüfung möglich ist und das ist bei XHTML der Fall.

Äußerlich fällt auf, dass alle Tags jetzt klein geschrieben sind (`<html>` statt `<HTML>`) und einzeln stehende Tags wie zum Beispiel `<BR>` jetzt abgeschlossen werden müssen: `<br />`. Für den Webdesigner ergibt sich eine viel bessere Überprüfungsmöglichkeit für die Syntax und eine praktisch vollkommene Trennung von Inhalt und Darstellung, weil die ursprünglich in HTML enthaltenen Formatier-Kommandos wie zum Beispiel `<FONT>` oder `<B>` jetzt nicht mehr verwendet werden und Style-Sheets an ihre Stelle treten. Für professionelle Webseiten ist jedenfalls XHTML vorzuziehen. Um eine Seite auf XHTML-Korrektheit zu überprüfen, verwendet man den Syntax-Checker von W3C. *http://validator.w3.org/*

Beispiele

Auf den vorigen zwei Seiten findet man Beispiele für die grundlegende Syntax von SVG.

Eindrucksvoll ist der Linux-Pinguin Tux, der auch auf der Titelseite zu dieser Ausgabe verwendet wird.

Die beiden letzten Grafiken zeigen, dass man SVG bestens für Diagramme verwenden kann.

Bei der Darstellung der Schriften (35.svg und 36.svg) zeigen sich große Abweichungen bei der Darstellung in verschiedenen Browsern. Wahrscheinlich ist es erforderlich, die Attribute noch genauer festzulegen, um die Freiheitsgrade bei der Darstellung einzuschränken.

SVG-Viewer für Internet-Explorer

Um auch am Internet-Explorer SVG-Dateien direkt anzeigen zu können, benötigt man ein Plug-In, das man hier downloaden kann:

http://www.adobe.com/svg/viewer/install/main.html

Hier kann man den Svg-Viewer testen:

http://www.adobe.com/svg/viewer/install/svgtest.html

Wenn das Plug-In erfolgreich installiert ist, sieht man folgendes Bild:

Leider wird dieses Plugin von Adobe nicht mehr unterstützt. Es ist daher nur als ein Behelf zu verstehen. Die Versionsnummer dieses Viewer ist 3, die aktuelle Version, der laufenden Adobe-Entwicklung aber bereits 6.

Wer laufend mit SVG arbeiten will, sollte Opera, Chrome oder Firefox verwenden. Diese Browser machen den besten Eindruck. Man kann auch in die Bilder hineinzoomen, natürlich verlustfrei, denn es handelt sich ja um Vektorgrafiken.

SVG-Editoren

Für einfache Versuche kann man den SVG-Kode so wie jede Html-Datei mit einem Texteditor herstellen. Doch SVG ist nicht so fehlertolerant wie Html. Schreibfehler führen zu Fehlermeldungen.

Um daher den Kode händisch herstellen und bearbeiten zu können, braucht man einen Grafik-Editor. Diese Editoren helfen beim Entwurf der statischen Teile eines Entwurfs. Verbreitete Programme zur Herstellung von SVG-Kode sind zum Beispiel Adobe Illustrator, Corel Draw oder GIMP. Diese benutzen aber SVG nicht als primäres Format sondern nur als eines von vielen Export/Import-Formaten. Der entstehende SVG-Kode ist etwa so gut wie der HTML-Kode einer im HTML-Format gespeicherten Word-Datei.

Die im folgenden Abschnitt beschriebenen Editoren sind dagegen auf die Herstellung von SVG-Grafiken spezialisiert. Der entstehende SVG-Kode ist sehr effizient.

Die Schwierigkeiten bei dem Datenaustausch zwischen diesen reinen SVG-Editoren und etwa CorelDraw entstehen durch die verschiedenartige Philosophie dieser Programme.

Wenn man eine einfach formatierte SVG-Datei in CorelDraw importiert danach wieder exportiert, wird der Kode zum Teil stark verändert. Zu den geringfügigen Veränderungen gehört, dass jedes Tag eine ID bekommt und dass die Koordinaten auf die verwendete Seitengröße A4 umgerechnet werden. Bei komplexeren Strukturen kann der Kode aber bis zu Unkenntlichkeit verändert werden. Das liegt daran, dass CorelDraw sein eigenes internes Format in SVG umrechnet und sich dabei auf die einfachsten Strukturen beschränkt. Beim Export in einen reinen SVG-Editor spielt das weniger Rolle, wohl aber beim Import, denn wenn der SVG-Editor das SVG-Format voll nutzt, versagt CorelDraw, denn die komplexe SVG-Kodierung wird von CorelDraw falsch interpretiert.

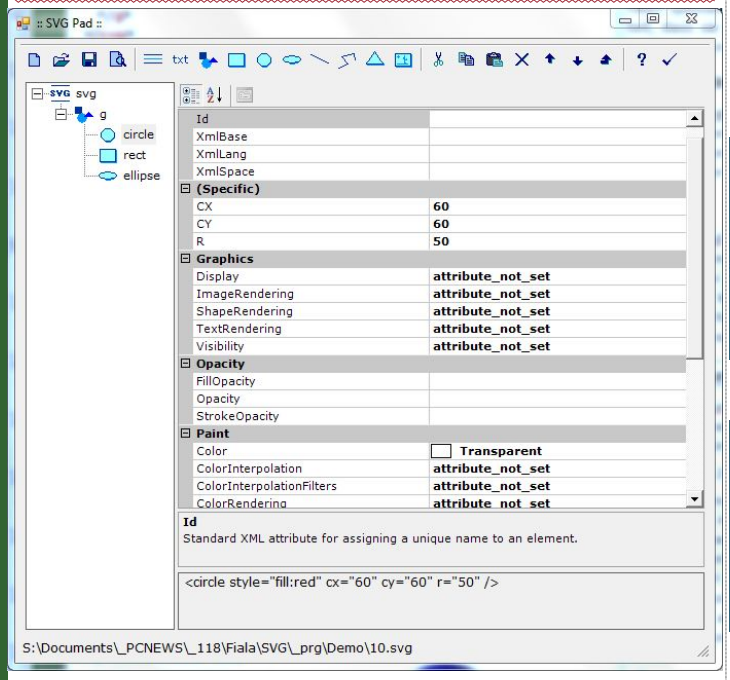
Man muss es sich daher zur Regel machen, möglichst auf die fortgeschrittenen Kodierungstechniken zu verzichten und beschränkt sich auf die einfacheren Strukturen.

SVG-Pad

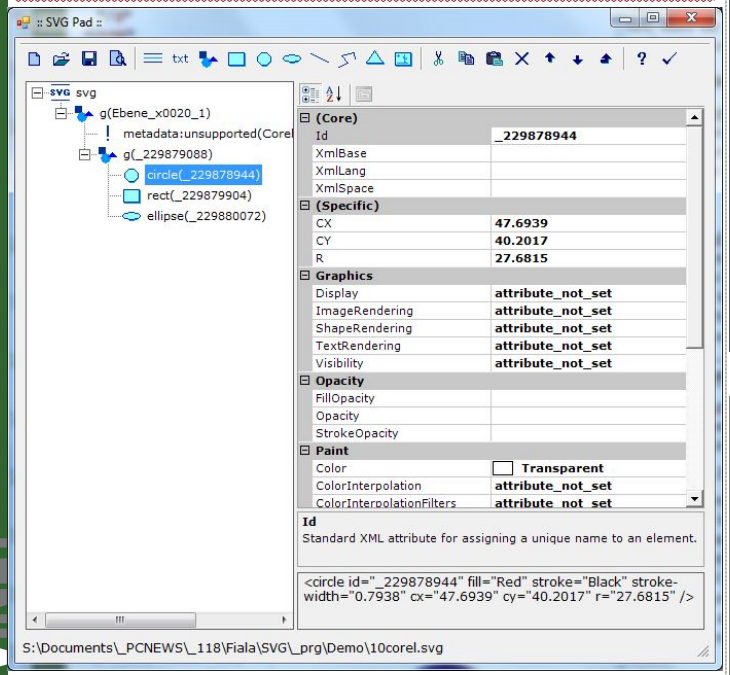
Mit SVG-Pad editiert man eine SVG-Datei auf dem Tag/Attribut-Niveau. Das Programm gibt einen Überblick über die verfügbaren Elemente in SVG und bei jedem Element werden nur die zulässigen Attribute angezeigt. Das Programm verfügt aber über keine Darstellungsmöglichkeit. Man speichert den generierten Kode und zeigt ihn mit einem Browser an.

Nachteil des SVG-Pad: es werden nur die grundlegenden Typen unterstützt. Bei komplexeren Typen bekommt man den Hinweis „not supported“. Wenn man dagegen SVG erlernen will, ist diese Darstellung seh angenehm, weil sie praktisch das Handbuch erspart.

Datei 10.svg im Editor SVG-Pad



BDatei 10.svg nach Import in und Export aus CorelDraw



AKs SVG-Editor

Dieser Editor ist ein Javascript-Programm und läuft in einem Fenster des Internet-Explorers ab. Das installierte Adobe-Plug-In ist erforderlich. Andere Browser sind für die Darstellung nicht geeignet.

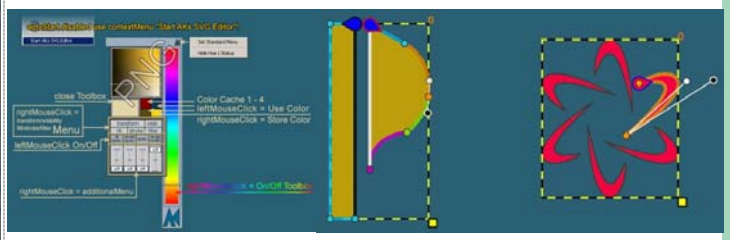
http://www.krammel.net/.

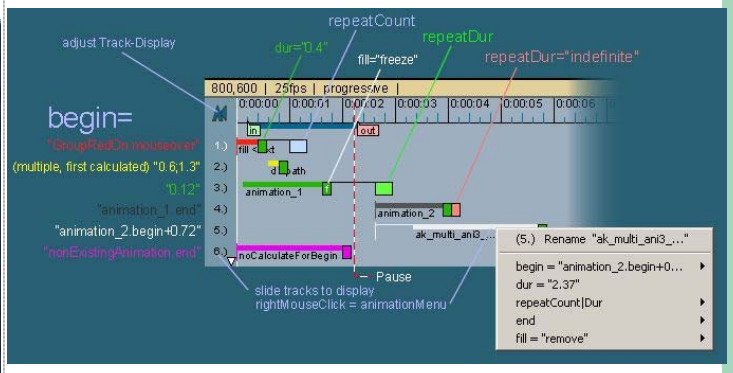
Dieser Editor ist noch „nah am SVG-Kode“. Es ist aber erstaunlich, was aus dem einen MB Kode herausgeholt werden kann. Besonders eindrucksvoll ist die Möglichkeit, Animationen mit SVG-Grafik herzustellen.

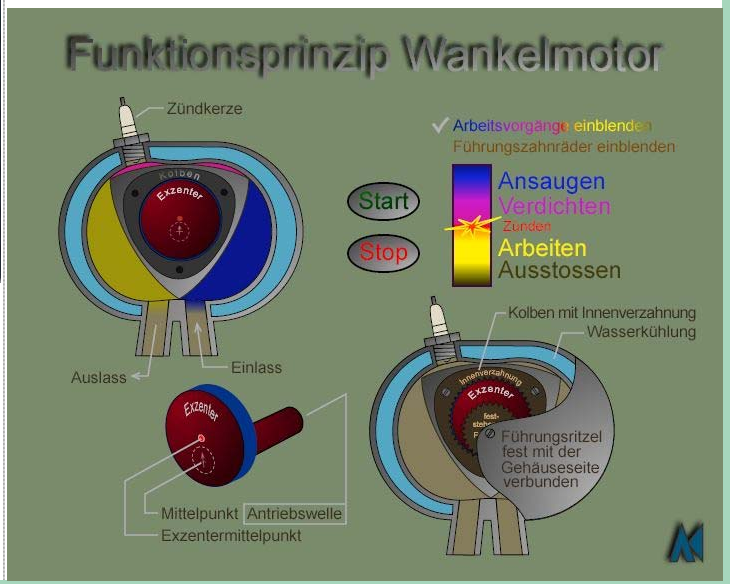
Hier einige Eindrücke aus der Hilfedatei:

Nachteilig erscheint mir diese enge Bindung an den Internet-Explorer, noch dazu, weil gerade dieser Browser seinerseits von einem nicht mehr unterstützen Adobe-Plugin abhängt. Der entstehende SVG-Kode läuft nur am Internet-Explorer.

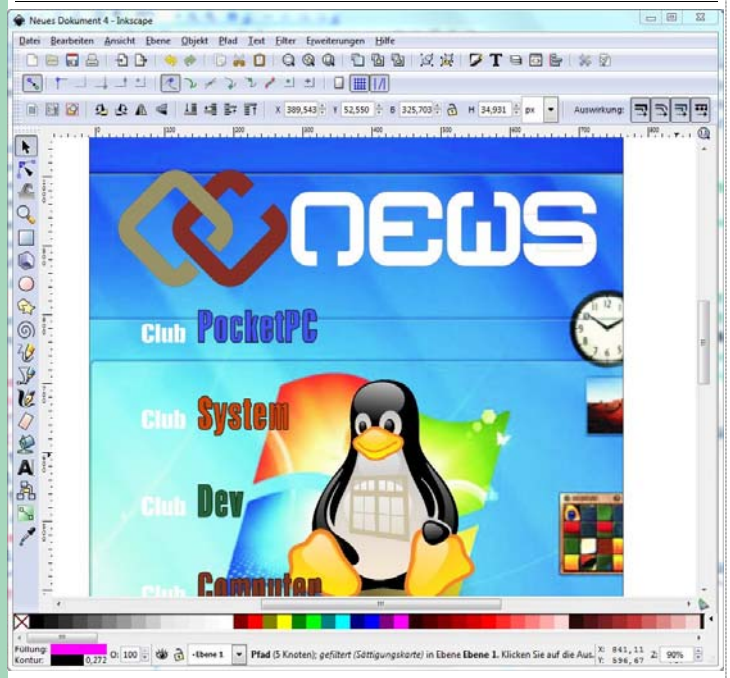
Der Grund dürfte aber die Spezialisierung auf Animationen sein. Das letzte Bild zeit die Animation eines Wankelmotors, das Bild darüber zeigt die Einstellungen für das Drehbuch.







Inkscape



Es war sehr einfach, die PCNEWS-Coverseite in Incscape zu übernehmen. Wirklich vermisst habe ich keine Funktion aus CorelDraw. Im Gegenteil! Es gibt zahlreiche weitergehende und sehr nützliche Funktionen, die auch in den kommerziellen Programmen nicht verfügbar sind. Eindrucksvoll die überaus große Zahl von Filtern, die man auf Objekte anwenden kann.

Mathematiklehrer werden sich freuen, dass man die besonderen Punkte eines Dreiecks für beliebige Dreiecke automatisch einzeichnen lassen kann.

In vielen Details kann man die große Nähe zwischen dem Format SVG und den möglichen Einstellungen erkennen.

Homepage

http://www.inkscape.org/?lang=de

Anleitungen

http://inkscapetutorials.wordpress.com/

http://de.wikipedia.org/wiki/Inkscape

http://de.wikibooks.org/wiki/Inkscape

http://wiki.inkscape.org/

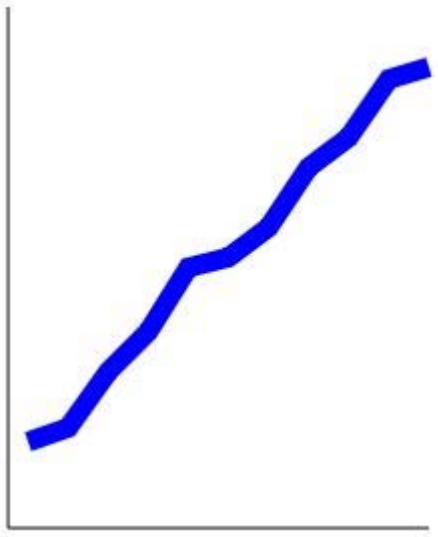
Beispiel aus der Messtechnik

Eine Messkurve bestehend aus x- und y-Werten ist darzustellen. Dimension und Achsenbeschriftung lassen wir der Einfachheit halber außer Acht. Die Werte sind (jeweils x- und y-Wert).

0 3,3 - 1 4 - 2 6,8 - 3 8,8 - 4 12 - 5 12,5 - 6 14 - 7 17 - 8 18,5 - 9 21,4 - 10 22

Um aus diesen Zahlenpaaren ein Diagramm herzustellen, müssen die Messwerte in die geeignete SVG-Form umgewandelt werden. Zunächst muss man beachten, dass die Zählung der y-Achse von oben nach unten verläuft. Weiters muss man eine geeignete Skalierung wählen. In unserem Beispiel zeichnen wir zwei Achsen als Linien und die Kurve als Mehrfachlinie dazu. Jeder Messwert wird mit 10 multipliziert (damit wird er ganzzahlig), außerdem wird 10 addiert, um Platz für eine Achsenbeschriftung zu bekommen. Die x-Achsenwerte werden mit dem Faktor 2 multipliziert.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
    width="500" height="500">
 <g transform="translate(100,100)">
   <line x1="10" y1="310" x2="220" y2="310" stroke="black"
      stroke-width="1" />
   <line x1="10" y1="310" x2="10" y2="50" stroke="black"
      stroke-width="1" />
   <polyline fill="none" stroke="blue" stroke-width="10"
      points="20,267 40,260 60,232 80,212 100,180 120,175
              140,160 160,130 180,115 200,86 220,80" />
 </g>
</svg>
```



Man kann einen Linienzug durch ein `line`-Tag oder ein `polyline`-Tag darstellen. Das `polyline`-Tag hat aber gegenüber der einfachen Linie den Vorteil, dass dickere Linien an den Knickstellen ordentlich verbunden werden und nicht abgehackt erscheinen. Das `g`-Tag gruppiert die Elemente und verschiebt das Bild mit dem Attribut `translate`. (Auch Drehungen, und Verzerrungen sind möglich.) Dazu kommt noch der unveränderliche Dokumentenkopf mit der Angabe der Darstellungsgröße 500, 500. Hinweis: Sowohl die Blattgröße als auch die Positionswerte in den einzelnen Grafikelementen können auch als Dezimalzahl in mm, cm und Points angegeben werden.

Programmatische Herstellung von SVG-Kode

Bei veränderlichen Werten wie zum Beispiel bei Messwerten oder bei Börsenkursen ist es nicht möglich, Seiten händisch zu editieren; vielmehr muss ein Programm die variablen Daten mit den statischen Bestandteilen verbinden. Eine kleine Unbequemlichkeit ist die Komplexität der SVG-Beschreibungssprache, und dass es eben XML-Kode ist, der gegenüber Schreibfehlern intolerant ist. Es empfiehlt sich daher, den Kode nicht selbst herzustellen, etwa in der Form

```
string s = "<circle cx=\""+cx+"\" cy=\"" + cy +"\" style=....
```

sondern dazu eine Bibliothek zu benutzen, die die einzelnen Objekte in Klassen formuliert und man lediglich die entsprechenden Objekte zu generieren hat und den String `s` als Ergebnis abrufen kann. Eine solche Bibliothek kommt von Ucancode (auch Hersteller von SVG-Pad):

Bibliothek zur Erstellung der SVG-Datei

Die folgende Bibliothek ist in C# geschrieben und ist für die Verwendung mit Visual Studio vorgesehen.

http://www.ucancode.net/ActiveX_Programming_Article_Com_Tutorial/Create-Open-Load-Save-Import-Export-Edit-SVG-File-Document-With-C-NET-Class-Library.htm

Mit dieser Bibliothek kann man die Grafikelemente bequem generieren. Man muss lediglich die dll-Datei in den Ordner `bin` kopieren und als Verweis zu dem aktuellen Projekt hinzufügen. Im eigentlichen Programm generiert man das obige Beispiel so (die Klassenelemente und Funktionsnamen wurde hier der Einfachheit halber weggelassen):

Bild.aspx

```
using SVGLib; // Einbingung der Bibliothek

// Erzeugt den Dokumentenkopf
// eines Svg-Dokuments
SvgDoc doc = new SvgDoc();

// Erzeugt das Root-element svg
// mit den dazugehörigen Attributen
SvgRoot root = doc.CreateNewDocument();
root.Width = "500";  // Zeichenbreite
root.Height = "500";  // Zeichenhöhe

// x-Achse
SVGLib.SvgLine x = new SVGLib.SvgLine(
doc, "10", "310", "220", "310", Color.Black);
l.StrokeWidth = "10";
l.Stroke = Color.Black;
doc.AddElement(root, x);

// y-Achse
SVGLib.SvgLine y = new SVGLib.SvgLine(
doc, "10", "310", "10", "50", Color.Black);
l.StrokeWidth = "10";
l.Stroke = Color.Black;
doc.AddElement(root, y);
```

```
// Messwerte
SVGLib.SvgPolyline p = new SVGLib.SvgPolyline(doc, "20,267 40,260
60,232 80,212 100,180 120,175 140,160 160,130 180,115 200,86
220,80");
poly_power.StrokeWidth = "10";
poly_power.Stroke = Color.Blue;
doc.AddElement(root, p);

// Datei speichern
bool FehlerFrei =
doc.SaveToFile(@"C:\pfad\Bild.svg");
if (FehlerFrei)
  Response.Write("Datei gespeichert");
else
  Response.Write("Datei nicht gespeichert");
```

Als Ergebnis wird die Datei `Bild.svg` im Pfad `C:\pfad` gespeichert.

Der Vorteil dieser Vorgangsweise ist, dass man sich nur mehr um die eigentlichen Variablen kümmern muss, denn die XML-Tags und die Namen der Attribute werden durch die einzelnen Klassen gekapselt. Es ist jetzt keine Schwierigkeit mehr, statt der konstanten Werte in diesem Beispiel die Variablen aus der Tabellenkalkulation, der Datenbank oder eben direkt aus der Messung einzufügen.

Umwandlung von SVG-Dateien Pixel-Grafik

Das war aber erst der halbe Weg, denn wir haben jetzt zwar eine SVG-Datei und diese Datei kann man mit den meisten Browsern anzeigen, doch wir wollen ja alle Browser bedienen und dazu braucht man ein Bild, wenigstens so lange als der Internet Explorer das SVG-Format nicht verstehen will. Kurioserweise stellt gerade eine Microsoft-OpenSource-Community ein Umwandlungprogramm zur Verfügung (*http://www.codeplex.com/svg*): eine C#-Bibliothek, die eine SVG-Datei als Input akzeptiert und daraus eine Grafikdatei (gif, jpg oder png) herstellt.

Um diese Bibliothek in der Entwicklungsumgebung zu nutzen, genügt es, die Dll in das Verzeichnis `bin` zu kopieren und danach diese Datei als Verweis zum Projekt hinzuzufügen. Nur, wenn man in den Source-Kode eingreifen will oder muss, kopiert man die Quelldateien in ein eigenes Verzeichnis und kompiliert es mit dem eigenen Projekt mit.

In meinen bisherigen Projekten erwies sich diese Bibliothek als nahezu perfekt.

Wie setzt man diese Bibliothek ein?

Im obigen Beispiel streicht man die Zeilen ab

```
bool FehlerFrei = doc.SaveToFile(@"C:\pfad\Bild.svg");
```

und erzeugt keine SVG-Datei. Stattdessen speichert man den generierten XML-Kode in einem String. Leider akzeptiert die Microsoft-Bibliothek keinen String als Input und erfordert stattdessen einen Stream. Meine Methode, den Stream zu erzeugen ist wahrscheinlich nicht optimal, das müsste viele einfacher gehen, aber egal, es funktioniert. Die eigentliche Umwandlung in eine Bild erfolgt durch die Befehlsfolge `Open() -> Draw() -> Save()`. Damit wäre das Bild schon erzeugt.

```
string s = doc.GetXML();
byte[] buffer = new byte[s.Length * 2];
for (int i = 0; i < s.Length; i++)
{
  b = char.ConvertToUtf32(s, i);
  [2 * i] = (byte)(b / 256);
  [2 * i + 1] = (byte)(b % 256);
}
MemoryStream memStream = MemoryStream(buffer);
SvgDocument.Open(memStream).Draw().Save(Path + "Bild.jpg",
ImageFormat.Jpeg);
string RelativePath = Request.Path;
Response.Flush();
RelativePath = RelativePath.Substring(0,
RelativePath.LastIndexOf(@"/"));
string PictureFileName = RelativePath + "Bild.jpg";
Image_Statistik.ImageUrl = PictureFileName;
```

Zum Abschluss wird noch der relative Pfad zu dem Bild ermittelt und die Bildadresse im Image-Element eingestellt.

Anwendung

In einer Html-Seite verwendet man

```
<img src="Pfad/Bild.jpg">
```

Wenn man das Bild gar nicht als Datei benötigt, kann man es durch folgende Zeile direkt in den Output-Stream schreiben:

Bild1.aspx

```
Response.ContentType = "image/jpeg";
SvgDocument.Open(memStream).Draw().
Save(Response.OutputStream, ImageFormat.Jpeg);
```

Diese Version ist Dynamik pur, denn das Bild existiert gar nicht als Datei; weder im SVG-Format noch im JPG-Format. Bei großen Grafiken kann aber die Berechnung zu einer unzulässigen Verzögerung führen. Wenn sich daher die Daten nicht gerade im Sekundentakt sondern zum Beispiel täglich oder wöchentlich ändern, sollte man die JPG-Datei speichern. Man regeneriert die JPG-Datei immer nur dann, wenn sich die Daten ändern.

Verwendete Tools und Dateien

- Visual Web Developer 2008 Express Edition
- Einen Microsoft-Server mit installiertem Framework, Version 3.5 (wichtig)
- Bibliothek von Ucancode zur Erzeugung des SVG-Kode
- Bibliothek von Microsoft zur Generierung eines Bildes aus dem SVG-Kode
- Musterdatei Bild.aspx erzeugt die Dateien Bild.svg und Bild.jpg
- Musterdatei Bild1.aspx erzeugt direkt ein Jpg-Bild

Für die Entwicklung wird kein Webserver benötigt, denn auf jedem Windows-PC in ein baugleicher Webserver für die Entwicklung vorhanden *(Verwaltung->Internet-Informationsdienste)*. Man muss nur darauf achten, dass das Framework Version 3.5 installiert ist, denn die Microsoft-Bibliothek erfordert ein besonderes Modul LINC, das nur dort enthalten ist.

Bild ohne Programm herstellen

Nun, es geht auch einfacher. Im Zuge des Testens wurde ein WebTool entwickelt, das für kleine Anwendungen auch von anderen Usern benutzt werden kann: *http://iam.at/svg2jpg/*

Wenn man eine SVG-Datei herstellt, egal ob mit einem Zeichenprogramm oder mit einem serverseitigen Programm (ASP, ASPX oder PHP) genügt es mit der folgenden Anwendung, die Adresse *http://iam.at/svg2jpg/* aufzurufen und man erhält als Antwort das zugehörige Bild in die eigene Seite geliefert.

Der konkrete Aufruf erfolgt so:

http://iam.at/svg2jpg?url=<Internet-Adresse der SVG-Datei>

url kann eine beliebige Adresse auf Ihrem eigenen Server oder sonst wo im Internet sein.

Beispiel:

http://iam.at/svg2jpg?url=http://iam.at/svg2jpg/exa/Beispiel1.svg

Anwendung in einer HTML-Datei

In einer HTML-Datei ist ein Vektor-Grafik-Bild im Rahmen eines `img`-Tag einzufügen. Der Grafik-Kode befindet sich auf der Adresse *http://iam.at/svg2jpg/exa/Beispiel1.svg* (wenn Sie das in Ihrer Anwendung verwenden, steht hier: *http://MeineDomäne.at/Pfad/MeineSvgDatei.svg*).

```
<img src="http://iam.at/svg2jpg?url=http://iam.at/exa/svg2jpg/
Beispiel1.svg">
```

Es ist aber zu beachten, dass der eigene Server Dateien mit der Endung svg auch tatsächlich als Text zurückschickt. Beim IIS ist das normalerweise nicht der Fall. Daher muss bei den Mime-Typen des Servers die Dateiendung svg bekannt gemacht werden und mit dem Typ `text/plain` verknüpft werden. (Clubmitglieder, die ihren Webspace mit *helm.ccc.at* verwalten, können diesen Eintrag unter dem Menüpunkt *MIME-Typen* hinzufügen.)

```
.svg    image/svg
```

Wenn man keine Möglichkeit hat, den Eintrag vorzunehmen, dann muss man die SVG-Datei in eine Textdatei umbenennen, damit die Datei vom Server korrekt behandelt wird und ohne Änderung an den Client zurückgeschickt wird.

Weitere Beispiele zum Ausprobieren finden Sie im Verzeichnis http://iam.at/svg2jpg/exa/.

SVG ist eine Sprache, die sich laufend entwickelt. Es kommt daher vor, dass manche fortgeschrittene Techniken nicht funktionieren, etwa die Datei

http://iam.at/svg2jpg/?url=http://iam.at/svg2jpg/exa/23.svg

Zusammenfassung

Es muss im Interesse des Anwenders sein, dass seine Arbeiten in Formaten gesichert werden, die auf einem möglichst breiten Konsens vieler Hersteller beruhen und nicht von einem einzigen Erzeuger abhängen. HTML, XML und SVG sind solche Formate. Proprietäre Formate können bestenfalls als Übergangslösung akzeptiert werden.

Gerade für das SVG-Format gibt es mit Incscape auch ein den großen Vektor-Grafik-Programmen gleichwertiges Äquivalent das dieses Format als zentrales Darstellungsformat verwendet.

Auf der Browserseite zwingt der normalerweise sehr üppig ausgestatte Internet-Explorer durch die SVG-Verweigerung zu komplizierten Lösungen. Der Entwicklungsaufwand für portable Anwendungen wird sehr groß, wie man an diesem Beispiel sieht. Erst in der Version 9 des Internet -Explorers gibt es zaghafte Versuche, SVG zu implementieren (ca. 25%).

Links zu diesem Artikel

Alle Dateien zum Download befinden sich bei der Webversion dieses Artikels.

Die Umwandlung einer SVG-Datei in JPG-Format kann man hier vornehmen:

Download	http://pcnews.at/pcn/1xx/11x/118/002400/_prg
Demo	http://fiala.member.pcc.ac/
Webdienst	http://iam.at/svg2jpg/

Programm zum Umwandeln eine SVG-Datei in eine JPG-Datei. Erfordert die Bibliotheken svg.lib und svglib.dll

Programmkode

svg2jpg.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="svg2jpg.aspx.cs" Inherits="svg2jpg" %>
```

sv2jpg.aspc.cs

```
using System;
using Svg;
using SVGLib;
using System.Drawing.Imaging;
using System.IO;
using System.Net;
using System.Text;
using System.Drawing;
public partial class svg2jpg : System.Web.UI.Page
{
  void Fehlermeldung(string Message)
  {
    SVGLib.SvgDoc doc = new SvgDoc();
    SVGLib.SvgRoot root = doc.CreateNewDocument();
    root.Width = "300";
    root.Height = "100";
    SVGLib.SvgRect Background = new SVGLib.SvgRect(doc, "0", "0",
"300", "100", "0", Color.White, Color.White);
    doc.AddElement(root, Background);
    SVGLib.SvgText FehlerText = new SVGLib.SvgText(doc);
    FehlerText.Value = Message;
    FehlerText.X = "20";
    FehlerText.Y = "20";
    FehlerText.FontFamily = "Arial Black";
    FehlerText.FontSize = "15pt";
    doc.AddElement(root, FehlerText);
    string s = doc.GetXML();
    byte[] buffer = new byte[s.Length];
    for (int i = 0; i < s.Length; i++)
    {
      buffer[i] = (byte)s[i];
    }
    MemoryStream memStream = new MemoryStream(buffer);
    Response.ContentType = "image/jpeg";
    SvgDocument.Open(memStream).Draw().Save
(Response.OutputStream, ImageFormat.Jpeg);
    Response.End();
  }
  bool IsFile(string FileName)
  {
    if (FileName.IndexOf(".") > 0)
      return true;
    else
      return false;
  }
  string ReadUri(string Path)
  {
    Uri uri;
    try
    {
      uri = new Uri(Path);
    }
    catch
    {
      Fehlermeldung("Falsches Url-Format\n" + Path);
      return "";
    }
    string[] s = uri.Segments;
    int SegmentCount = s.Length;
    if (IsFile(s[SegmentCount - 1]))
      SegmentCount--;
    WebRequest wReq = WebRequest.Create(Path);
    WebResponse wResp;
    try
    {
      wResp = wReq.GetResponse();
    }
    catch
    {
      Fehlermeldung("Dokument nicht gefunden\n" + Path);
      return "";
    }
    Stream respStream = wResp.GetResponseStream();
    StreamReader reader = new StreamReader(respStream,
Encoding.Default);
    String respHTML = reader.ReadToEnd();
    respStream.Close();
    return respHTML;
  }
  protected void Page_Load(object sender, EventArgs e)
  {
    string Url = Request.QueryString["url"];
    if (Url == null) Url = "";
    // Response.Write("Url:" + Url + "<br>"); Response.Flush();
    if (Url == "")
      Fehlermeldung("Kein Url angegeben\n...?url=");
    string s = ReadUri(Url);
    byte[] buffer = new byte[s.Length];
    for (int i = 0; i < s.Length; i++)
    {
      buffer[i] = (byte)s[i];
    }
    MemoryStream memStream = new MemoryStream(buffer);
    Response.ContentType = "image/jpeg";
    try
    {
      SvgDocument doc = new SvgDocument();
      SvgDocument.Open(memStream).Draw().Save
(Response.OutputStream, ImageFormat.Jpeg);
    }
    catch
    {
      Fehlermeldung("Grafik-Fehler");
    }
  }
}
```