Mit CD

Datenqualität + + + IT-Architektur + + + Iterative Softwareentwicklung S&S

entwickler magazin

# entwickler magazin

Software, Systems & Development

4.08 | Juli / August | www.entwickler-magazin.de | Deutschland € 6,50 Österreich € 7,00 Schweiz sFr 13,40



## Testing

**Sicherer, schneller und effizienter testen**

## IBM Data Studio

**Vereinfachte Entwicklung für DB2**

## So cute!

**Plattformunabhängiges Development mit Qt**

## Kreative Silverlight-Anwendungen

**Dynamischer Zugriff mit dem Downloader-Objekt**

## CD-Inhalt

### Software

Qt Open Source Edition for C++ Developers
Webmin 1.420
WordPress 2.5.1
Xalan C++ 1.10 und Xalan Java 2.7.1
Virtual Treeview component

### Testversionen

▶ Advantage Database Server 9.0
▶ Conzept 16

▶ Alle Infos auf Seite 5

Datenträger enthält nur Lehr- oder Infoprogramme

4 194156 606509 04

## Einsatz des Downloader-Objekts in Silverlight-Anwendungen

# Silberlichtverstärker

**Ab Silverlight 1.0 steht in Form des Downloader-Objekts eine vielseitige Technik für den dynamischen Datenzugriff bereit. Damit lassen sich Anwendungen realisieren, die sich bei externen Diensten oder Datenquellen bedienen und Inhalte bei Bedarf nachladen können. Dieser Beitrag vermittelt die programmiertechnischen Grundlagen auf JavaScript-Basis.**

**von Thomas Meinike**

Jegliche Technologien für RIA (Rich Internet Applications) machen von dynamischen Datenzugriffen auf externe Ressourcen Gebrauch, um ihre Benutzer an den viel gepriesenen „neuartigen Anwendungserfahrungen" teilhaben zu lassen. Was immer man darunter verstehen mag. AJAX ist ein weiteres Buzzword im Bereich Webentwicklung mit dem *XMLHttpRequest*-Objekt als technologischem Hintergrund. Dieses Objekt ging aus dem ursprünglich von Microsoft eingeführten ActiveX-Objekt *XMLHTTP* hervor und wird auch ab Internet Explorer 7.0 unterstützt. Die Weiterentwicklung erfolgt mittlerweile unter W3C-Federführung [1]. Insofern liegt die Annahme nahe, dass auch Silverlight diesen Mechanismus vorsieht. Bei näherer Betrachtung folgt dieser Annahme die Erkenntnis: Im Prinzip ja, aber mit dem Downloader-Objekt doch ein wenig anders.

### Voraussetzungen

Da dieser Beitrag keine Einführung in Silverlight 1.0 darstellt, werden die Installation des Plug-ins selbst, das Silverlight-SDK

[2] und Grundkenntnisse der Syntax von XAML (Extensible Application Markup Language) sowie Erfahrungen im Umgang mit JavaScript vorausgesetzt. Die Minimalkonfiguration einer Silverlight-Anwendung demonstriert ein kürzlich veröffentlichtes Video [3]. Somit beziehen sich die folgenden Ausführungen auf den in einem XHTML-Dokument eingebetteten Silverlight-Container, der den jeweiligen XAML-Inhalt darstellt.

```
<!-- im head-Element -->
<script src="silverlight.js" type="text/javascript">
                                                </script>
<!-- ggf. weitere script-Elemente mit eigenen Funktionen -->

<!-- im body-Element -->
<div id="SilverlightPlugInHost">
 <script type="text/javascript">createSilverlight();
                                                </script>
 <!-- XAML-Datei wird in dieser Funktion als source
                                        referenziert -->
</div>
```

Mittels passenden Event-Handlern wie *Loaded* oder *MouseLeftButtonDown* lassen sich eigene JavaScript-Funktionen aus dem XAML-Kontext heraus ansteuern. Auf der vom Autor betriebenen Plattform zum Thema Silverlight stehen die im Folgenden behandelten Beispiele im Quellcode und direkt lauffähig zur Verfügung [4].

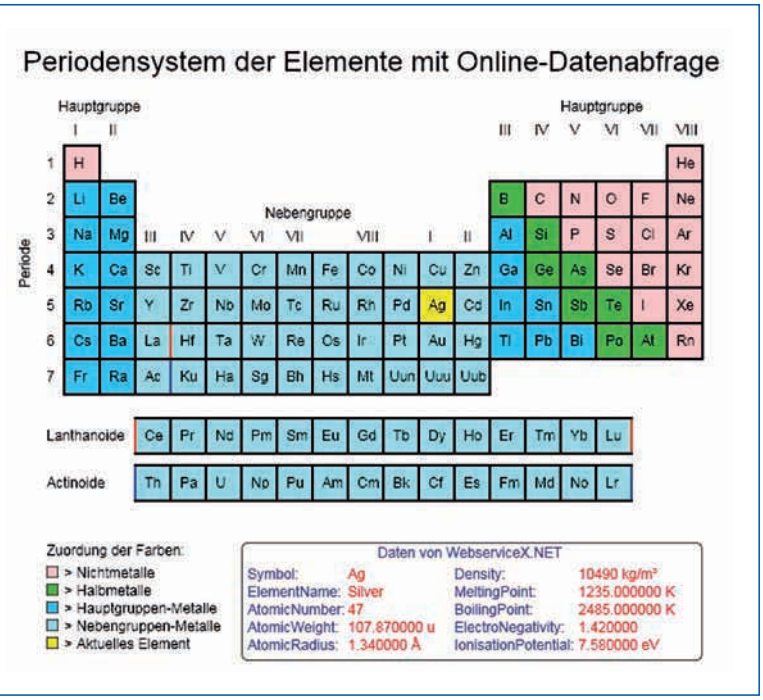
## Das Downloader-Objekt

Dieses Objekt ist für den asynchronen Zugriff auf externe Ressourcen unter Verwendung der *GET*-Methode konzipiert und aus dieser Sicht mit *XMLHttpRequest* vergleichbar. Es werden jedoch einige abweichende Eigenschaften, Ereignisse und Methoden verwendet. Eine neue Downloader-Instanz kann direkt aus dem XAML-Kontext heraus erzeugt werden (z. B. über <*Canvas … Loaded="Datenzugriff"*>).

```
function Datenzugriff(sender,eventArgs)
{
 var sl_ctrl=sender.getHost();
 var dl=sl_ctrl.createObject("downloader");
 // weiterer Code …
}
```

Die Variable *sl_ctrl* enthält eine Referenz auf den aktuellen Silverlight-Container

Abb 1.: Silverlight-Version des Periodensystems der Elemente




und wird später noch zum Einfügen dynamisch erzeugter Inhalte benötigt, während in der Variable *dl* die Referenz auf das neue Downloader-Objekt abgelegt ist. Wesentliche Methoden sind *abort()*, *getResponseText()*, *open()* und *send()*. Zu den häufig benötigten Eigenschaften gehören *downloadProgress*, *responseText*, *status*, *statusText* und *uri*. Hinzu kommen noch die Ereignisse *Completed*, *DownloadFailed*, *DownloadProgressChanged*.

Neben den ausgewählten Merkmalen enthält die MSDN-Beschreibung weitere [5]. Eine Anmerkung zur Schreibweise: Im .NET-Umfeld wird üblicherweise die *UpperCamelCase*-Notation verwendet, während die JavaScript-Syntax eher in *lowerCamelCase* notiert wird. Insofern favorisiert der Autor bei der Kodierung von Eigenschaften und Methoden letztere, etwa *send()* statt *Send()* oder

**Listing 1**

```
<?xml version="1.0" encoding="UTF-8"?>
<Canvas xmlns="http://schemas.microsoft.com/
                                        client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/
                                        2006/xaml"
    Loaded="Serverdaten">

 <TextBlock x:Name="ausgabe" Text="…"
                    Canvas.Left="30" Canvas.Top="30"
      FontFamily="Arial" FontSize="18"
                                Foreground="#090"/>

</Canvas>
```

**Listing 2**

```
function Serverdaten(sender,eventArgs)
{
 var dl=sender.getHost().createObject("downloader");

 dl.addEventListener("DownloadFailed",function
                                        (sender,eventArgs)
 {
  sender.findName("ausgabe").text="Fehler bei der
                                        Datenabfrage!";
 });

 dl.addEventListener("Completed",function
                                        (sender,eventArgs)
 {
  if(sender.status==200)sender.findName("ausgabe").
                                text=sender.responseText;
  else sender.findName("ausgabe").text="Fehler bei der
                                        Datenabfrage!";
 });

 dl.open("GET","text.php");
 dl.send();
}
```


Abb 2.: Die Daten stammen aus einem XML-Dokument

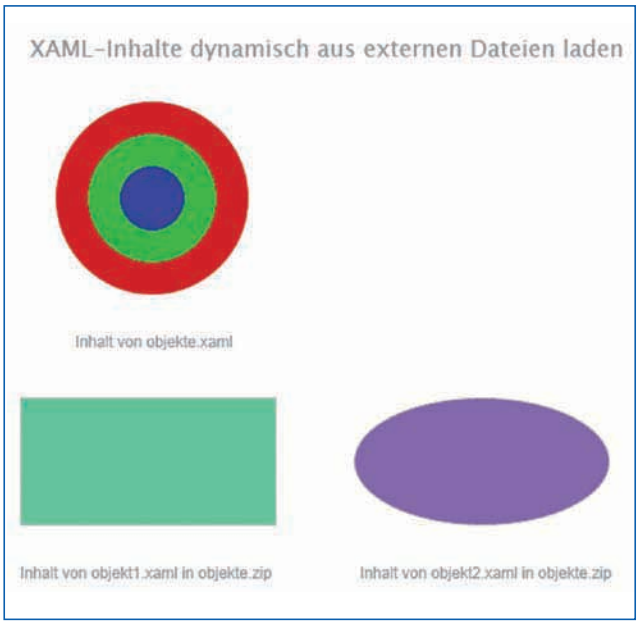



Abb. 3: Aus Dateien eingebundene XAML-Inhalte


Abb. 4: Ansicht der aus einem ZIP-Archiv geladenen Bilder


Abb. 5: Mit externem Font formatierter Text

*findName()* statt *FindName()*. Dieses Vorgehen erscheint legitim, da auch im MSDN nicht konsequent unterschieden wird und in der Praxis bisher keine Probleme mit der gewählten Schreibweise auftraten. Bei der XAML-Auszeichnung besteht diese Wahlfreiheit natürlich nicht mehr.

## Serverdaten verarbeiten

Der erzeugten Downloader-Instanz lassen sich die gewünschten Ereignisse zuweisen und daran Funktionen koppeln (hier in anonymer Schreibweise formuliert).

```
dl.addEventListener("Completed",function
                                    (sender,eventArgs)
{
 alert(sender.responseText);
});

dl.addEventListener("DownloadFailed",function
                                    (sender,eventArgs)
{
 alert("Fehler!");
});
```

Schließlich löst die Abfolge der Methoden *open()* und *send()* den Datenzugriff aus.

```
dl.open("GET","einfach.txt");
dl.send();
// Ende der Funktion Datenzugriff()
```

Es wird eine einfache Textdatei abgeholt und deren Inhalt bei Erfolg (*Completed*) über *responseText* in einer Meldungsbox ausgegeben. Schlägt der Download hingegen fehl (*DownloadFailed*), erscheint die definierte Fehlermeldung. Damit liegt bereits eine erste funktionsfähige Anwendung vor. In den Listings 1 bis 3 wird das gezeigte Prinzip etwas komfortabler demonstriert, d.h. die Ausgaben werden direkt in die XAML-Struktur übernommen. Zudem werden die Informationen serverseitig von einem PHP-Skript generiert und auch der HTTP-Statuscode 200 (OK) wird ausgewertet (Beispiel *server_daten.xaml*).

Statt aus der Eigenschaft *responseText* können die gelieferten Inhalte auch durch den Methodenaufruf *getResponse-*

*Text("")* mit Leerstring als Argument beschafft werden.

## Ladefortschritt messen und anzeigen

Über das bisher noch nicht verwendete Ereignis *DownloadProgressChanged* kann insbesondere der Ladezustand größerer Downloads verfolgt werden. Eine daran geknüpfte Funktion wird periodisch aufgerufen und kann den erreichten Fortschritt an der Eigenschaft *downloadProgress* ablesen. Ihr Wert bewegt sich im Intervall von 0 bis 1 und wird in Schritten von 0.05 (5 %) erhöht. Mit dieser Information lässt sich ein Fortschrittsbalken unter Verwendung eines *Rectangle*- und eines mittig darüber platzierten *TextBlock*-Elements umsetzen. Durch stetiges Neusetzen des Attributs *Width* ändert sich die Rechteckbreite bis zum Maximalwert, während der erreichte Prozentwert über das *TextBlock*-Attribut *Text* ausgegeben wird. Im später thematisierten Beispiel zum Nachladen von Bildern kommt dieser Code zum Einsatz:

```
dl.addEventListener("DownloadProgressChanged",
                                    function(sender,eventArgs)
{
 var pr=Math.floor(sender.DownloadProgress*100);
 pr_rect_obj.width=pr*4.3; // maximale Rechteck-
                           //          Breite: 430
 pr_proz_obj.text=pr+"%";
});
```

## XML-Daten einbeziehen

Naiv (oder aus AJAX-Sicht praktisch) gedacht, sollte sich zu *responseText* eine Eigenschaft *responseXML* gesellen. Das ist beim Downloader-Objekt leider nicht der Fall. Eine der ersten Ideen des Autors für eine dynamische Silverlight-Anwendung war die Umsetzung des bereits mit SVG-Techniken realisierten Periodensystems der chemischen Elemente auf der Basis eines Web Service [6, 7]. Die gelieferten XML-Daten wurden zunächst in ein JSON-Format [8] umgeformt und anschließend über *responseText* in den JavaScript-Kontext überführt und an *eval()* zur Auswertung übergeben. Für das Element Silber ergibt sich diese Datenstruktur, deren rechts stehenden Werte über *TextBlock*-Elemente ausgegeben werden (vgl. Abbildung 1 zum Beispiel *periodensystem.xaml*).

```
{
'AtomicNumber' : '47',
'ElementName' : 'Silver',
'Symbol' : 'Ag',
'AtomicWeight' : '107.87',
'BoilingPoint' : '2485',
'IonisationPotential' : '7.58',
'ElectroNegativity' : '1.42',
'AtomicRadius' : '1.34',
'MeltingPoint' : '1235',
'Density' : '10490'
}
```

Dennoch muss nicht auf XML-Verarbeitung verzichtet werden. Dazu ist es erforderlich, die als Text übertragene XML-Quelle in ein Document-Objekt umzuwandeln. Da hier das Silverlight-Plug-in selbst nicht behilflich ist, werden browserspezifische DOM-Methoden benötigt. Listing 4 zeigt die kompakte Funktion *XMLDocument()*, die als Argument den Inhalt von *sender.responseText* innerhalb einer Downloader-Aktion übernehmen kann. Das als *xmldoc* gelieferte Objekt kann mit konventionellen DOM-Methoden wie *getElementsByTagName()* weiterverarbeitet werden. Im Beispiel *xml_verarbeitung*.

### Listing 3

```
header("Content-Type: text/plain; charset=UTF-8");
print "Dieser Text wurde serverseitig mit PHP erzeugt\n".
   "und über ein JS-Downloader-Objekt eingefügt.";
```

### Listing4

```
function XMLDocument(xmldata)
{
 var xmldoc=null;

 if(window.ActiveXObject)
 {
  xmldoc=new ActiveXObject("Microsoft.XMLDOM");
  xmldoc.async=false;
  xmldoc.loadXML(xmldata);
 }
 else if(window.DOMParser)
 {
  xmldoc=(new DOMParser()).
parseFromString(xmldata,"text/xml");
 }

 return xmldoc;
}
```

### Listing 5

```
<?xml version="1.0" encoding="UTF-8"?>
<Canvas xmlns="http://schemas.microsoft.com/
                                        client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/
                                        2006/xaml">

 <Ellipse Canvas.Left="50" Canvas.Top="70"
              Width="150" Height="150" Fill="#C00"/>

 <Ellipse Canvas.Left="75" Canvas.Top="95"
              Width="100" Height="100" Fill="#0C0"/>

 <Ellipse Canvas.Left="100" Canvas.Top="120"
              Width="50" Height="50" Fill="#00C"/>

 <TextBlock Canvas.Left="65" Canvas.Top="250"
                                        Foreground="#999"
  FontFamily="Arial" FontSize="12"
                    Text="Inhalt von objekte.xaml"/>

</Canvas>
```

### Listing 6

```
<?xml version="1.0" encoding="UTF-8"?>
<files info="Bilddateien in fotos.zip">
 <file>foto_01.png</file>
 <file>foto_02.png</file>
 <file>foto_03.png</file>
 <file>foto_04.png</file>
</files>
```

### Listing 7

```
<!-- Fortschrittsbalken -->
<Rectangle Canvas.Left="20" Canvas.Top="560"
Width="0" Height="25" Stroke="#EEE"
StrokeThickness="0.5" RadiusX="12.5" RadiusY="12.5"
x:Name="pr_rect">
 <Rectangle.Fill>
  <LinearGradientBrush>
   <GradientStop Color="#DFF" Offset="0.0"/>
   <GradientStop Color="#7AC" Offset="0.8"/>
   <GradientStop Color="#589" Offset="1.0"/>
  </LinearGradientBrush>
 </Rectangle.Fill>
</Rectangle>

<!-- Fortschrittswert in Prozent -->
<TextBlock Canvas.Left="220" Canvas.Top="564"
Width="430" Height="25" Foreground="#00C"
FontFamily="Lucida" FontSize="12" Text="%"
x:Name="pr_proz"/>
```

*xaml* werden die erhaltenen Daten tabellarisch aufgelistet (Abbildung 2).

## XAML- und ZIP-Inhalte verarbeiten

Es wurde bereits auf die Möglichkeit des direkten Zugriffs auf Textdateien hingewiesen. Auch XAML-Dateien sind grundsätzlich textbasiert und können ohne separates Parsen als XML in bereits bestehende XAML-Inhalte eingefügt werden, sofern die Wohlgeformtheit erhalten bleibt. Im Beispiel *xaml_verarbeitung1.xaml* wird in ein bestehendes *Canvas*-Element mit dem Attribut *x:Name="container"* der komplette Inhalt von *objekte.xaml* eingefügt (Listing 5).

Nach dem erfolgreichen Übertragen folgt die Übergabe des erhaltenen XAML-Codes an die Methode *createFromXamlDownloader()*, die den erhaltenen Datenstrom direkt in ein verwendungsfähiges Objekt umwandelt. Dieses erreicht sein Ziel unterhalb des bereits bestehenden *Canvas*-Elements durch die Anwendung der *add()*-Methode.

```
var xaml_obj=sl_ctrl.content.createFromXamlDownloader
                                        (sender,"");
if(xaml_obj=="Canvas")cont_obj.children.add(xaml_obj);
else ... // Ausgabe einer Fehlermeldung
```

Auch die Formulierung *createFromXaml(sender.responseText)* wäre hier möglich, wobei diese sich eher bei direkter Zuweisung von XAML-Code als Zeichenkette anbietet. An Stelle der beiden Anführungszeichen können auch die Namen von Dateien stehen, die aus einem ZIP-Archiv stammen. Im Beispiel *xaml_verarbeitung2.xaml* wird mit *dl.open("GET","objekte.zip")* ein solches Archiv vom Server abgeholt. Die ohne weiteres Zutun extrahierten Inhalte *objekt1.xaml* (Rechteck) und *objekt2.xaml* (Ellipse) erscheinen in analoger Weise gemäß Abbildung 3 auf der Zeichenfläche. Somit lassen sich Ressourcen kompakt vorhalten und bei Bedarf in die Anwendung einbinden.

## Sonstige Medieninhalte

Dieses Vorgehen ist auch für Bild-, Audio- und Videodateien attraktiv. Im Beispiel *zip_verarbeitung.xaml* wird die Datei *fotos.zip* geladen und der aus vier PNG-Grafiken bestehende Inhalt dargestellt (foto_01.png bis foto_04.png). Zugewiesen werden die Dateien entweder der *Source*-Eigenschaft eines *Image*-Objekts oder wie im Beispiel der *ImageSource*-Eigenschaft eines *ImageBrush*-Objekts. Letzteres kann als Füllung von Grundformen verwendet werden. Im Beispiel kommen *Rectangle*-Elemente mit abgerundeten Ecken zum Einsatz, wobei für die eigentliche Übergabe die Methode *setSource()* zuständig ist.

```
var rectcode='<Rectangle ...><Rectangle.Fill><ImageBrush
ImageSource=""/></Rectangle.Fill></Rectangle>';
var rect_obj=sl_ctrl.content.createFromXaml(rectcode);
// ...
rect_obj.fill.setSource(sender,"foto_01.png");
```

Da sich der Inhalt der ZIP-Datei nicht zur Laufzeit bestimmen lässt, ist die genannte Nummerierung der Bilddateien mit einem festen Stamm (foto_??.png) sinnvoll, sodass sich die Dateinamen über eine Schleife zuweisen lassen. Ein anderer praktikabler Ansatz besteht im zusätzlichen Ablegen einer beschreibenden XML-Datei zu den im ZIP-Archiv enthaltenen Dateien (Listing 6), die vor dem Bildzugriff wiederum mit der Funktion *XMLDocument()* ausgewertet wird. Abbildung 4 zeigt die geladenen Bilder und Listing 7 den verwendeten Aufbau des oben bereits genannten Fortschrittsbalkens mit zu den Bildern passendem Farbverlauf. Audio- bzw. Video-Dateien können ebenfalls über *setSource()* dem dafür bestimmten *MediaElement* zugewiesen werden.

Eine weitere Stärke des Downloader-Objekts ist der Umgang mit nicht unmittelbar unterstützten Fonts in den Formaten True Type und Open Type.

```
dl.open("GET","YanoneTagesschrift.ttf");
// ... in Completed-Funktion:
ausgabe.setFontSource(sender);
ausgabe.fontFamily="Yanone Tagesschrift";
ausgabe.fontSize=48;
ausgabe.text="Yanone Tagesschrift";
```

Neben der Methode *setFontSource()* ist die Kenntnis des exakten Namens der Schriftart entscheidend, die der Eigenschaft *fontFamily* zuzuweisen ist. Sollte Bedarf zum Zurücksetzen des Fonts auf dessen Voreinstellung bestehen, kann *setFontSource(null)* am entsprechenden Textobjekt aufgerufen werden. Abbildung 5 veranschaulicht das im Beispiel *fonts_extern.xaml* umgesetzte Prinzip mit einer Schriftart, die unter einer Creative-Commons-Lizenz veröffentlicht wurde [9]. Das mit dem hinzu geladenen Font belegte *TextBlock*-Element hat diese Ausgangsform:

```
<TextBlock x:Name="ausgabe" Text="..." Canvas.Left="30"
                                    Canvas.Top="60" Foreground="#090"/>
```

## Fazit

Das Downloader-Objekt ist ein unverzichtbares Hilfsmittel zur kreativen Entwicklung von Silverlight-Anwendungen mit dynamischem Zugriff auf unterschiedliche Medieninhalte und Daten. Ausgehend von den beschriebenen Beispielen sollte es nicht schwer fallen, diese als Basis für eigene Ideen zu verwenden und mit den dargestellten Techniken zu experimentieren. Hinweis: Alle Beispiele funktionieren auch mit der ersten Beta-Version von Silverlight 2.0.

***Dr. Thomas Meinike*** *ist seit 1997 an der Hochschule Merseburg (FH) als Lehrkraft tätig. Seine Arbeitsschwerpunkte sind XML-Anwendungen in der Technischen Dokumentation, Onlinehilfen und Webentwicklung. Er verfasst regelmäßig Fachartikel und hält Vorträge zu Themen im XML-Umfeld. E-Mail-Kontakt: thomas.meinike@hs-merseburg.de*

## Links & Literatur

[1] W3C: www.w3.org/TR/XMLHttpRequest

[2] Microsoft Silverlight: silverlight.net/GetStarted

[3] Palermo, M.: Silverlight Minimum Steps; silverlight.net/learn/learnvideo.aspx?video=49075

[4] Meinike, T.: Silverlight/XAML – Learning by Coding; slxlbc.datenverdrahten.de

[5] MSDN: msdn.microsoft.com/en-us/library/bb979676.aspx

[6] Meinike, T.: Chemiestunde – Entwicklung von dynamischen Datenzugriffstechniken für Scalable Vector Graphics; XML & Web Services Magazin 1.2004, S. 42–44

[7] Periodic Table Web Service: www.webservicex.net/WS/WSDetails.aspx?WSID=19&CATID=7

[8] JavaScript Object Notation: www.json.org

[9] Font-Yanone-Tagesschrift: www.yanone.de/typedesign/tagesschrift