

Rich Internet Applications mit Adobe Flex 3

Wer sind wir?

Kerstin Götze



- Diplom-Medieninformatikerin (FH)
- J2SE, J2ME, AS2, AS3, Flex
- RockAByte GmbH

Daniel Bertram



- B.Sc., Medieninformatik
- Masterstudent
- AS3, Flex, Objective C
- RockAByte GmbH

Was erwartet uns heute?

- Flex-Überblick
- „Hello World“
- Entwicklungsumgebung
- Design View des Flex Builders
- MXML und AS3
- Online-Dokumentation
- Container und Controls
- Bilder und Schriften

Was ist Flex?

- Open Source RIA-Framework, bestehend u.A. aus:
 - wiederverwendbaren und erweiterbaren GUI Objekten
 - Event-Handling-Komponenten
 - Data-Handling-Komponenten
- Erzeugt SWFs, die durch Flash Player 9 (oder höher) oder Adobe AIR gerendert werden
- Clientseitige Technik
- AS3: Objektorientierte Programmiersprache
- MXML: XML basierte Markup Language für einfaches Erstellen des Anwendungs-Layouts
- Adobe AIR: Flex Framework für Desktop Anwendungen

Abgrenzung zu Flash

- Flash
 - richtet sich hauptsächlich an Designer
 - ist Timeline-basiert
 - Flash arbeitet nicht mit MXML
 - Flex erweitert das in Flash verwendete AS3-Framework durch viele nützliche Klassen und Komponenten
- Flex ist Flash für Programmierer :)

Übung 1:

Erste Schritte mit dem Flex Builder

„Hello Flex“

Ein Flex Projekt erstellen

- File > New > Flex Project
- Projektnamen und Speicherort wählen
- Typ: Web Application
- Server Type: None
- „Finish“ klicken
- Rechtsklick auf das Projekt > Preferences > Flex Compiler > Require Flash Player version: 10.0.0

Die Projektstruktur

- **src**
 - enthält den Source Code
- **libs**
 - enthält alle zusätzlich benötigten Bibliotheken
- **html-template**
 - enthält Template Dateien für den HTML Wrapper
- **bin-debug**
 - enthält Debug-Version der kompilierten Anwendung
- **bin-release**
 - enthält Release-Version der kompilierten Anwendung

Die Hauptprojektdatei

- MXML-Datei vom Typ `Application`
- muss zwingend vorhanden sein
- Basiscontainer und Startpunkt der Anwendung
- alle weiteren GUI Elemente werden diesem Container hinzugefügt

„Hello Flex“

```
<?xml version="1.0"?>
<!-- mxml\HelloWorld.mxml -->

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Label text="Hello Flex!" />

</mx:Application>
```

„Hello Flex“ erweitern

```
<?xml version="1.0"?>
<!-- mxml\HelloWorld.mxml -->

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:TitleWindow title="My Window">
        <mx:Label text="Hello Flex!"/>
    </mx:TitleWindow>

</mx:Application>
```

Die Design View

- kann nur für MXML Klassen aktiviert werden
- WYSIWYG (What You See Is What You Get) Editor der es erlaubt per Maus GUI Objekte in der Anwendung zu platzieren
- Das Aussehen der GUI Objekte kann per graphischer Benutzungsoberfläche angepasst werden
- entsprechender Code wird automatisch generiert

Alternativen zum FlexBuilder & Hilfsmittel

- Alternativen:
 - FlashDevelop
 - FDT Eclipse Plugin
 - beliebiger Editor
- Hilfsmittel
 - Apache Ant
 - Maven

15 min Erholungspause... :)

Online Dokumentation

- dokumentiert alle Klassen des Frameworks mit ihren
 - Eigenschaften
 - Methoden
 - Styles
 - Effekten
 - Events
 - Konstanten
- zeigt oft Code-Beispiele für die Verwendung der Klassen
- <http://livedocs.adobe.com/flex/3/langref/>

Online Hilfe

- umfangreiche Dokumentation und Erklärung der Komponenten und Konzepte von Flex, z.B.:
 - Erste Schritte
 - Benutzeroberflächen
 - Event Handling
 - Data Handling
 - Fortgeschrittene Techniken
 - Benutzung des FlexBuilders
- gibt Code-Beispiele
- http://livedocs.adobe.com/flex/3/html/help.html?content=Part2_DevApps_1.html

Nützliche Ressourcen

- **Component Explorer**
<http://examples.adobe.com/flex3/componentexplorer/explorer.html>
- **Style Explorer**
<http://examples.adobe.com/flex3/consulting/styleexplorer/Flex3StyleExplorer.html>
- **Flex Cookbook**
<http://www.adobe.com/cfusion/communityengine/index.cfm?event=homepage&productId=2>
- **Flex Developer Center**
<http://www.adobe.com/devnet/flex/>
- **FlexBox**
<http://flexbox.mrinalwadhwa.com/>

Übung 2:

Verwenden von Online Doku & Hilfe

Übung 2

Aufgabe: Findet mit Hilfe der Online Dokumentation Alternativen zur Verwendung der Label Komponente und fügt sie der Anwendung hinzu.

- **Verwendet nicht die Design View**
- **Tip: Nehmt das Paket `mx.controls` unter die Lupe**
- Zusatzaufgabe 1: Fügt der Anwendung fünf weitere Komponenten eurer Wahl hinzu.
- Zusatzaufgabe 2: Sucht in der Dokumentation nach einer Möglichkeit das TitleWindow per Mouse zu bewegen (Tip: nur ein spezielles Attribut muss gesetzt werden)

MXML & AS3

- beide Sprachen können in Flex Anwendungen miteinander kombiniert werden
- AS3 Methoden können in MXML Klassen eingefügt werden (Script Block), aber MXML Code nicht in AS3 Klassen
- alles, was mit MXML ausdrückbar ist, kann auch in AS3 geschrieben werden, umgekehrt gilt das nicht
- MXML wird hauptsächlich für das Platzieren, Gestalten und Verschachteln von Layout-Komponenten verwendet
- AS3 wird für die Programmierung der Anwendungslogik und des Event Handlings verwendet

MXML

- ein Tag besteht aus
 - Namespace
 - Klassenname
 - Attributen mit Wertzuweisungen
- Beispiel: `<mx:Button id="b1" label="click me"/>`
- Tags können verschachtelt werden
- Über den Wert des id-Attributs kann ein Objekt angesprochen werden
- Kommentare werden so erzeugt: `<!-- Kommentar -->`
- Das Root Tag einer Klasse definiert alle verwendeten Namespaces
`<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">`

AS3

- kann mittels eines `<Script>` Blocks in einer MXML Klasse eingefügt werden
- kann in Form einer kompletten AS3 Klasse verwendet werden
- Objektorientierte Programmiersprache
- Wichtigste Schlüsselwörter
 - `class`
 - `function`
 - `var`
- Kommentare werden so erzeugt:
 - `// Kommentartext`
 - `/* Kommentartext */`

AS3 Beispiel

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="onCreationComplete()"
>
    <mx:Script>
        <![CDATA[
            import mx.controls.Label;

            private function onCreationComplete():void
            {
                // create label and set text
                var l:Label = new Label();
                l.text = "Hello Flex";
                l.setStyle("fontSize", 12);

                // add label to application
                this.addChild(l);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Übung 3:

MXML & AS3

Übung 3

Aufgabe: Es liegen zwei Layoutskizzen vor. Anhand dieser sollen die dargestellten Komponenten umgesetzt werden.

Das Kalenderfenster sollte in MXML oder mit Hilfe des Design Views erstellt werden.

Das Konfigurationsfenster soll mit AS3 in einem Skriptblock erstellt werden.

- Zusatzaufgabe 1: am Schriftgrößen-Slider sollen in 8er-Schritten Skalenbeschriftungen angezeigt werden.
- Zusatzaufgabe 2: Tage, die in der Vergangenheit liegen, dürfen im DateChooser nicht auswählbar sein.

Layout Container

- Container sind GUI Objekte, die dazu dienen andere GUI Objekte (Kindobjekte) aufzunehmen
- Container kümmern sich, wenn nötig, um das Scrolling und positionieren ihre Kindobjekte nach einem bestimmten Schema (vertikal, horizontal, absolut)
- Container positionieren ihre Kindobjekte entweder absolut oder relativ zueinander
- können ineinander geschachtelt werden
- Alle Container finden sich im Paket `mx.containers`

Layout Container - Beispiele

- Nur absolute Positionierung (Position jedes Kindes muss explizit bestimmt werden)
 - Canvas
- Nur relative Positionierung
 - HBox (horizontale Anordnung)
 - VBox (vertikale Anordnung)
 - Grid (Anordnung in einer Matrix)
- absolute oder relative Positionierung
 - Panel
 - TitleWindow

Layout Container - Sonderfälle

- Einen Sonderfall bilden die Container
 - Accordion
 - TabNavigator
 - ViewStack
- Diese Container können als direkte Kindobjekte nur andere Container enthalten, keine sonstigen GUI Objekte
- Die Kindobjekte können ihrerseits dann allerdings beliebige GUI Objekte enthalten

Übung 4:

Layout Container

Übung 4

Aufgabe: Erstellt ein neues Projekt „FlexMediaPlayer“.

Die Applikation soll ein Fenster mit zwei Tabs mit den Titeln „3D Properties“ und „Rotationseffekt“ enthalten. Das Fenster selbst hat den Titel Control Panel.

Auf den Aufgabenblättern seht ihr zwei Screenshots, die den Inhalt der Tabs und dessen Anordnung zeigen. Baut diese Screenshots nach und nutzt die gerade vorgestellten Layout Container, um die dargestellte Anordnung der Controls zu erreichen.

Setzt bei allen Slidern `liveDragging` auf `true`

Übung 4

- Zusatzaufgabe 1: Erzeugt einen vierten Tab in dem sich ein Accordion befindet. Das Accordion hat drei Teile. In jedem Accordionteil befindet sich je ein Text, der es beschriftet.
- Zusatzaufgabe 2: Erzeugt einen fünften Tab in dem sich in einer 3x3 Matrix angeordnet die folgenden Zeichen befinden: X, X, O, X, O, X, X, O, O. Verwendet dazu die Komponente Grid.

Multimediaobjekte verwenden

- Bilder, Schriften, SWFs & Sounds können extern eingelesen werden oder in die Anwendung eingebettet werden
- **Vorteil der Einbettung**
 - eingebettete Objekte werden in das SWF kompiliert wodurch ohne Verzögerung auf sie zugegriffen werden kann
- **Nachteil der Einbettung**
 - Einbetten von Objekten vergrößert die Anwendung und damit die Ladezeit

Schriften & Bilder einbetten

- Bilder/SWFs sollten eingebettet werden wenn
 - es wichtig ist die Ladezeit des Bildes so gering wie möglich zu halten
- Schriftarten sollten eingebettet werden wenn
 - unsicher ist, ob die Schrift auf jedem Client Computer installiert ist
 - Effekte auf Komponenten mit Schrift angewendet werden sollen
 - Antialiasing verwendet werden soll
- Beim Einbetten einer Schrift müssen alle verwendeten Font Types (plain, bold, italic) einzeln eingebettet werden

Schrift & Bild: unterstützte Formate

- Bilder/Animationen:
 - PNG
 - JPG
 - GIF (aber kein AnimatedGIF)
 - SVG (nur eingebettet)
 - SWF
- Schrift:
 - TrueType (TTF)
 - OpenType (OTF)

Schriften verwenden: Beispiel

```
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  fontFamily="myFontFamily"
>
  <!-- embedding the font -->
  <mx:Style>
    @font-face {
      src: url("../assets/font/verdana.TTF");
      /*src: local("Impact");*/
      fontFamily: myFontFamily;
      advancedAntiAliasing: true;
    }
  </mx:Style>

  <!-- using the default font -->
  <mx:Text text="Hallo Flex [embedded font]"/>

  <!-- using a non-embedded font -->
  <mx:Text text="Hallo Flex [external font]" fontFamily="Arial" />

</mx:Application>
```

Bilder verwenden: Beispiele

MXML Beispiel:

```
<!-- embedded image -->
<mx:Image id="embeddedImg" source="@Embed('../assets/logo.png')"/>

<!-- external image -->
<mx:Image id="externalImg" source="assets/logo.png"/>
```

AS3 Beispiel:

```
[Embed(source="../assets/logo.png")] public var imgCls:Class;

private function useImage():void{
    // creating an instance of the embedded image and using it
    var imgObj:BitmapAsset = new imgCls() as BitmapAsset;
    var img:Image = new Image();
    img.source = imgObj;

    // creating an instance an image with external source
    var img2:Image = new Image();
    img2.source = "assets/logo.png";
}
```

Übung 5:

Bilder & Schriften einbetten

Übung 5

Aufgabe: Fügt das Flex Logo (flexlogo.png) als eingebettetes Bild in die rechte obere Ecke der Anwendung ein.

Bettet außerdem die euch zur Verfügung gestellte Schriftart in die Anwendung ein. Setzt sie als Standardschriftart für die Anwendung.

Zusammenfassung & Feedback

Flex Workshop - Tag 2

Was gestern geschah...

- Flex-Überblick
- „Hello World“
- Entwicklungsumgebung
- Design View des Flex Builders
- MXML und AS3
- Online-Dokumentation
- Container und Controls
- Bilder und Schriften

Fragen, Probleme oder
Anregungen?

Was erwartet uns heute?

- Klassen, Methoden & Variablen
- Getter und Setter
- Events, Event Listener & Event Handler
- Security Sandbox
- Audio & Video
- Styles, Skins & Filter
- lokaler Dateizugriff

Klassen

- Dienen der Strukturierung und Modularisierung der Anwendung
- Definieren Methoden und Eigenschaften, die alle Instanzen dieser Klasse besitzen
- Klassen können von anderen Klassen erben und/oder Interfaces implementieren
- Können als AS3 oder MXML Datei angelegt werden
- AS3 Klassen besitzen einen Konstruktor, der immer ausgeführt wird, wenn eine neue Instanz der Klasse erzeugt wird
- Name der Klasse = Dateiname = Name des Konstruktors

Klassen: MXML Beispiel

Klasse ButtonBox.mxml

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Klasse, die die Klasse HBox erweitert -->
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button
        id="button1"
        label="button1"
    />

</mx:HBox>
```

Klassen: AS3 Beispiel

Klasse ButtonBox.as

```
package customClasses
{
    import mx.containers.HBox;
    import mx.controls.Button;

    // Klasse, die die Klasse HBox erweitert
    public class ButtonBox extends HBox
    {
        private var _button1:Button;

        // Konstruktor
        public function ButtonBox()
        {
            _button1 = new Button();
            _button1.label = "button1";
            this.addChild(_button1);
        }
    }
}
```

Klassen: Verwendung

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:customClasses="customClasses.*"
  creationComplete="init()"
>

  <!-- Verwendung im MXML Block -->
  <customClasses:ButtonBox/>

  <mx:Script>
    <![CDATA[
      import customClasses.ButtonBox;

      // Verwendung im AS3 Code
      private function init():void{
        var buttonBox2:ButtonBox = new ButtonBox();
        this.addChild(buttonBox2);
      }
    ]]>
  </mx:Script>
</mx:Application>
```

Getter und Setter

- spezielle Methoden, um Eigenschaften eines Objektes zu lesen und zu schreiben
- werden wie Eigenschaften einer Klasse aufgerufen, nicht wie Methoden (Bsp.: `Player.volume = 0.9;`)
- **Achtung:** Getter und Setter einer Eigenschaft, dürfen nicht genau den gleichen Namen haben, wie diese Eigenschaft, sonst entsteht bei Aufruf des Getters oder Setters eine Endlosschleife
- spezielle Syntax in AS3:
 - `public function get volume():Number`
 - `public function set volume(value:Number):void`

Getter und Setter: Beispiel

```
public class Player
{
    private var _volume:Number;

    // Setter
    public function set volume(value:Number):void{
        _volume = value;
    }

    // Getter
    public function get volume():Number{
        return _volume;
    }
}
```

Getter und Setter: Verwendung

ActionScript 3:

```
public function changePlayerVolume():void
{
    private var player:Player = new Player();
    player.volume = 0.5;
}
```

MXML:

```
<local:Player
    volume="0.5"
/>
```

Übung 1:

Klassen, Getter & Setter

Übung 1

Aufgabe:

Erzeugt eine MXML-Komponente „ControlPanel“. Diese Soll von TitleWindow erben. Verschiebt alle GUI-Elemente des in den vorherigen Übungen erzeugten Fensters in diese Klasse.

Erzeugt eine weitere MXML-Komponente „MediaPlayer“, die von TitleWindow ableitet. Diese soll Getter und Setter für die Eigenschaft `_url` vom Typ String erhalten. Der Getter soll die aktuelle URL per `trace()` ausgeben. Fügt dem Fenster außerdem eine Instanz von `VideoDisplay` hinzu.

Erzeugt ein AS3-Klasse „Playlist“, die ebenfalls von TitleWindow ableitet. Setzt im Konstruktor Breite, Höhe und Titel des Fensters.

Übung 1

Aufgabe (Fortsetzung):

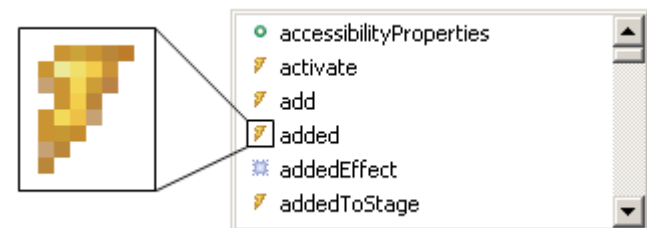
Fügt je eine Instanz aller drei Klassen der Applikation hinzu und setzt `_url` für die `MediaPlayer`-Instanz auf „../assets/video/demo.flv“.

Events

- Benachrichtigung über Ereignisse in der Anwendung
- können von Eingabegeräten oder von Objekten geworfen werden
- Benachrichtigung z.B. über
 - Benutzereingaben (z.B. MouseEvents, KeyboardEvents)
 - Veränderungen in der Benutzeroberfläche
 - Veränderung oder Eingang von Daten
 - Lebenszyklus eines Objektes

Events

- Können durch Listener abgefangen und mit einem Event Handler behandelt werden
- Neben den vorgefertigten Events können auch eigene Events (Custom Events) erstellt und geworfen werden
- Online Doku: Beschreibt für jedes Objekt, welche Events dieses werfen kann
- FlexBuilder: In der Liste der Code Completion sind Events mit folgendem Symbol dargestellt:



Event Listener & Handler

- Für ein beliebiges Objekt, kann auf Events gelauscht und reagiert werden, die dieses Objekt wirft
- An das Objekt, welches das Event wirft, wird ein Listener angefügt. Dieser definiert, auf welches Event gelauscht werden soll und welche Methode (Event Handler) beim Auftreten des Events ausgeführt werden soll.
- Im Event Handler können nun beliebige Aktionen als Reaktion auf das Event ausgeführt werden
- Wird ein Listener nicht mehr benötigt, sollte er entfernt werden
- Asynchrones / Nebenläufiges Verfahren

Event Listener & Handler

- Event Handler erhält als Parameter das ausgelöste Event
- Aus diesem Event können folgende Eigenschaften ausgelesen werden:
 - Event Typ
 - Objekt, welches das Event geworfen hat
 - Je nach Eventtyp noch weitere Informationen

Das creationComplete Event

- Event, das geworfen wird, wenn eine GUI Komponente sich vollständig aufgebaut hat, inklusive ihrer Kindobjekte
- erst dann kann sicher auf Eigenschaften des Objekts, wie Höhe und Breite und die Kindobjekte zugegriffen werden
- MXML Klassen besitzen keinen expliziten Konstruktor. Um in der `Application` Klasse den Einstieg in die Anwendung zu ermöglichen, bieten sich die Events `creationComplete` oder `applicationComplete` an

Event Listener & Handler: creationComplete Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="init()"
>
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;

            // creationComplete Handler
            private function init():void
            {
                Alert.show("application created");
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Event Listener & Handler: MXML Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Button
        id="myButton"
        label="click me"
        click="onButtonClick(event) "
    />
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;

            // Event Handler
            private function onButtonClick(e:MouseEvent):void{
                Alert.show("button clicked. event type: "+e.type);
            }
        ]]>
    </mx:Script>

</mx:Application>
```

Event Listener & Handler: AS3 Beispiel

```
private var myButton:Button;

private function createButton():void
{
    myButton = new Button();
    myButton.label = "click me";

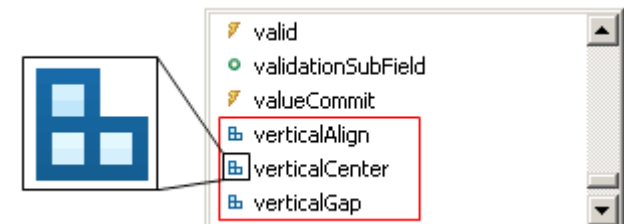
    // Listener hinzufügen
    myButton.addEventListener(MouseEvent.CLICK, onButtonClick);

    this.addChild(myButton);
}

// Event Handler
private function onButtonClick(e:MouseEvent):void
{
    // Listener entfernen
    myButton.removeEventListener(MouseEvent.CLICK, onButtonClick);
    Alert.show("button clicked");
}
```

Styles: Grundlagen

- dienen der individuellen Anpassung der Benutzungsoberfläche
- können für einzelne GUI Objekte oder ganze Klassen von GUI Objekten definiert werden
- können im Code oder per CSS (Cascading Style Sheet) gesetzt werden
- Online Doku: Beschreibt für jedes GUI Objekt welche Styles angepasst werden können und welche Default Werte gesetzt sind
- FlexBuilder: In der Liste der Code Completion sind Styles mit folgendem Symbol dargestellt:



Übung 2: Events

Übung 2

Aufgabe: Fügt dem Control Panel folgendes Verhalten hinzu:
Button „Einstellungen speichern“:

- Klick: Alert erscheint mit Text „Einstellungen gespeichert“
- Mouse über Button: Schrift des Buttons wird kursiv
- Mouse verlässt Button: Schrift des Buttons wieder normal

ColorPicker: Bei Farbauswahl, Hintergrundfarbe der Anwendung auf die gewählte Farbe setzen

Slider „Field of View“, Projektionsmittelpunkt X/Y“: Bei Slider-Bewegung neuen Wert mit trace()-Statement ausgeben

- Zusatzaufgabe: Über den „x“-Button in der Titelleiste des Control Panels, soll dieses geschlossen werden.

Security Sandbox

- Eine Flex-Anwendung läuft innerhalb einer sogenannten Security Sandbox, die unautorisierten Zugriff auf die Flex-Anwendung bzw. der Flex-Anwendung auf externe Ressourcen verhindert
- Folgende Regeln gelten:
 - Dateien/Anwendungen innerhalb der gleichen Sandbox können immer aufeinander zugreifen (Web Server: innerhalb der gleichen Domäne)
 - Dateien/Anwendungen in einer anderen Sandbox können nicht auf lokale Ressourcen (auf dem Client) zugreifen

Security Sandbox

- **Lokale Anwendungen**

- können entweder auf lokale Ressourcen zugreifen ODER auf Netzwerkressourcen
- soll auf beides zugegriffen werden können, muss die Anwendung manuell in die Trusted Zone des Flash Players aufgenommen werden

- **Anwendungen auf einem Server**

- können auf in der gleichen Domäne liegende Netzwerk- und lokale (auf dem Server liegende) Ressourcen zugreifen
- bei Ressourcen in einer anderen Domäne, muss dort eine crossdomain.xml liegen, die den Zugriff erlaubt

crossdomain.xml - Beispiele

```
<!-- allow access from every domain on every port -->
<cross-domain-policy>
  <allow-access-from
    domain="*"
    to-ports="*"
  />
</cross-domain-policy>
```

```
<!-- allow access only from specific domain on specific ports -->
<cross-domain-policy>
  <allow-access-from
    domain="*.myserver.com"
    to-ports="80,443,8100,8080"
  />
</cross-domain-policy>
```

Sandbox Typen

- abhängig von der Kompileroption `-use-network` und davon, ob die Anwendung lokal oder über das Netzwerk geladen wird, verändert sich der Sandbox Typ der Anwendung

<code>-use-network=</code>	geladen von	Sandbox Typ
<code>false</code>	lokal	local-with-filesytem
<code>true (Default)</code>	lokal	local-with-network
<code>true (Default)</code>	Server	remote
<code>false</code>	Server	nicht verfügbar / Fehler

Audio & Video

- unterstützte Audioformate: MP3
- unterstützte Videoformate: FLV
- Klasse **Sound**
 - Audiodatei wird automatisch gestreamt
 - ab Flash Player 10 kann Sound auch aus einem ByteArray Objekt gelesen werden
- Klasse **VideoDisplay** unterstützt
 - Progressive Download über HTTP
 - Streaming von Flash Media Server
 - Streaming von einer angeschlossenen Kamera

Übung 3:

Audio & Video

Übung 3

Aufgabe: Erweitert die Klasse MediaPlayer so, das der Setter für `_url` die Eigenschaft `source` für das VideoDisplay setzt.

Fügt außerdem folgende Komponenten mit entsprechender Funktionalität hinzu:

- Einen 'Start' Button
- Einen 'Stopp' Button
- Einen 'Pause' Button

Fügt einen Slider hinzu, der die Lautstärke des Videos regelt.

Zusatzaufgabe: Fügt einen weiteren Slider hinzu, mit dem ihr in dem Video navigieren könnt.

Styles & Skins

- **Styles** beziehen sich auf Eigenschaften wie Farben, Linienstärken, Transparenz usw.
- **Skins** beziehen sich auf visuelle Elemente eines Objektes, die durch Bitmaps, SWFs oder Klassen (Programmatic Skins), die Methoden zum Zeichnen von Vektorformen enthalten dargestellt werden
- **Skins** werden also durch Objekte beschrieben, während **Styles** nur einfache Werte annehmen können

Styles & Skins

- können für Klassen per CSS oder mit einem `<Style></Style>` MXML Block gesetzt werden
- können für konkrete Instanzen einer Klasse in MXML Syntax oder in AS3 mit der Methode `setStyle()` gesetzt werden
- einzelnen Objekten können über die Eigenschaft `styleName` in CSS definierte Styleklassen zugewiesen werden
- **Hinweis:** Flex CSS Syntax ist nicht zu 100% identisch mit der W3C CSS Syntax

Styles & Skins: Beispiele

MXML:

```
<!-- es wird der Style textAlign gesetzt sowie die Skins overSkin  
in Form einer Skin-Klasse und downSkin in Form eines PNGs -->  
  
<mx:Button  
    label="button styled via MXML"  
    color="0xFFFFFFFF"  
    overSkin="{MyProgrammaticButtonSkin}"  
    downSkin="@Embed(source='downSkin.png') "  
/>
```

AS3:

```
[Embed(source="../../../assets/downSkin.png")] public var imgCls:Class;  
  
private function setButtonStyles():void  
{  
    myButton.setStyle("color", "0xFFFFFFFF");  
    myButton.setStyle("overSkin", MyProgrammaticButtonSkin);  
    myButton.setStyle("downSkin", imgCls);  
}
```

externes CSS: Beispiel

Styles.css

```
Button {  
    fontSize: 15;  
    fillColors: #FF0000, #FF0000;  
}  
  
.greenButtonStyle {  
    fontSize: 12;  
    fillColors: #00FF00, #00FF00;  
}
```

CSS einbinden:

```
<mx:Style source="../../../assets/Styles.css"/>  
  
<!-- Button with default style -->  
<mx:Button label="default red Button" />  
  
<!-- Button with custom style -->  
<mx:Button label="green Button" styleName="greenButtonStyle" />
```

Filter

- können auf GUI Objekte angewandt werden, um z.B. Schatten, Leuchten, Weichzeichnung usw. zu erzeugen
- vordefinierte Filter im Paket `flash.filters`

Filter: Beispiele

MXML Beispiel:

```
<mx:Label id="label1" text="Filter Test" fontSize="20">  
  <mx:filters>  
    <mx:DropShadowFilter blurX="5" blurY="5" distance="20"/>  
  </mx:filters>  
</mx:Label>
```

AS3 Beispiel:

```
public function createGlowFilter():void  
{  
    var f:GlowFilter = new GlowFilter(0xFFFFFFFF);  
    var myFilters:Array = label1.filters;  
    myFilters.push(f);  
    label1.filters = myFilters;  
}
```

Übung 4:

Styles, Skins & Filter

Übung 4

Aufgabe: Passt die Anwendung mittels externem CSS und Filtern folgendermaßen an:

- Hintergrundfarbe der Anwendung: #ADCCA7
- Buttons allgemein: Farbe #7B9177, Alpha 1
 - Pause Button: Farbe #FFD91C
 - Stopp Button: Farbe #FF361C
- Slider: slider.png als Skin für den Anfasser
- Mediaplayerbuttons: GlowFilter in ihrer jeweiligen Farbe

Zusatzaufgabe: Erstellt eine Programmatischen Skin für den Pause Button. upSkin: Farbe #FFD91C, overSkin: Farbe #FFA14F, 2px breiter Rand in der Farbe #FF6100

Lokaler Dateizugriff

- ab Flash Player 10 kann direkt auf das lokale Dateisystem zugegriffen werden um Dateien zu laden oder zu speichern
- dazu wird die Klasse **FileReference** verwendet, die folgende Methoden zur Verfügung stellt: **browse()**, **load()**, **save()**, **download()**, **upload()**

Lokaler Dateizugriff: Beispiel

```
public function browseFilesystem() {  
    var file:FileReference = new FileReference();  
    file.addEventListener(Event.SELECT, selectHandler);  
    var filter:FileFilter =  
        new FileFilter("Images", "*.jpg;*.gif;*.png");  
    file.browse([filter]);  
}  
  
private function selectHandler(event:Event):void {  
    var file:FileReference = FileReference(event.target);  
    trace("selectHandler: name=" + file.name);  
}
```

Übung 5: Lokaler Dateizugriff

Übung 5

Aufgabe:

Fügt dem MediaPlayer einen Button „Video speichern“ hinzu.
Bei Klick auf diesen Button soll das Video von folgender URL
heruntergeladen und gespeichert werden:

http://www.rockabyte.com/kgoetze/flex_workshop/demo.flv

Zusammenfassung & Feedback

Flex Workshop - Tag 3

Was gestern geschah...

- Klassen, Methoden & Variablen
- Getter und Setter
- Events, Event Listener & Event Handler
- Security Sandbox
- Audio & Video
- Styles, Skins & Filter
- lokaler Dateizugriff

Fragen, Probleme oder
Anregungen?

Was erwartet uns heute?

- Data Provider
- Data Binding
- XML Parsing
- Web Service
- Effekte & Animationen
- Nativer 3D-Support
- Evaluation & Zertifikat

Data Provider

- Eigenschaft von listenbasierten GUI Komponenten, die ein Set von Datenobjekten anzeigen, z.B.
 - List
 - Tree
 - ComboBox
- Ist das zugewiesene Datenset vom Typ `ArrayCollection`, werden Änderungen (z.B.: Einfügen oder Löschen von Datenobjekten) direkt in der Benutzeroberfläche angezeigt, ohne dass ein manuelles Update erfolgen muss
- Datenobjekte im DataProvider können beliebig komplex sein, müssen aber einem bestimmten Schema folgen

Data Provider: einfaches Beispiel

```
<mx:Button label="add data" click="onButtonClick()" />
<mx>List id="list" width="400" height="100%" />

<mx:Script> <![CDATA[
    import mx.collections.ArrayCollection;

    private var dp:ArrayCollection;
    private var itemIndex:int=0;

    private function initDataProvider():void{
        dp = new ArrayCollection();
        dp.addItem("First Item");
        dp.addItem("Second Item");
        list.dataProvider = dp;
    }

    private function onButtonClick():void{
        dp.addItem("New Item" + itemIndex++);
    }
}]></mx:Script>
```

Data Provider: komplexes Beispiel

```
<mx:List id="list" width="400" onItemClick="onItemClick()" />
<mx:TextArea id="output" />

<mx:Script> <![CDATA[
    import mx.events.ListEvent;
    import mx.collections.ArrayCollection;

    private var dp:ArrayCollection;

    private function initDataProvider():void{
        dp = new ArrayCollection();
        dp.addItem({label:"Google", data:"www.google.de"});
        dp.addItem({label:"Yahoo", data:"www.yahoo.de"});
        list.dataProvider = dp;
    }

    private function onItemClick():void{
        output.text = list.selectedItem.data as String;
    }
}]></mx:Script>
```

Data Binding

- Binden von Daten eines Objektes (Source) an ein anderes Objekt (Target)
- Ändern sich die Daten des Source Objektes, wird das Target Objekt benachrichtigt und führt automatisch ein Update durch
- Anwendungsbeispiele: Binden von
 - Eigenschaften von Oberflächenelementen an andere GUI Elemente
 - Ergebnisse von Web Service Anfragen an Objekte
 - Datensets an listenbasierte GUI Elemente (Data Provider)
 - XML Daten an Objekte

Data Binding

- **Voraussetzungen** für das Data Binding:
 - zu bindende Variable muss über Getter & Setter les- und schreibbar sein, die mit dem Metadaten-Tag `[Bindable]` gekennzeichnet sind
- MXML: wird einem Attribut eine Variable zugewiesen, die „bindable“ ist (siehe Voraussetzungen) erfolgt automatisch ein Binding, ansonsten handelt es sich nur um eine einfache einmalige Wertzuweisung
- AS3: `var text:String = myObject.title` stellt nur eine einfache Wertzuweisung dar. Das Binding muss explizit mit Hilfe der Klasse `BindingUtils` gesetzt werden

Data Binding: MXML Beispiel

```
<!-- Binds the text from the input field to a non editable text
field -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- TextInput.text is readable/writable by a Getter &
    Setter -->
    <mx:TextInput id="myTI"/>                                <!--source-->
    <mx:Text id="myText" text="{myTI.text}"/>                <!--target-->

</mx:Application>
```

Data Binding: AS3 Beispiel

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:TextInput id="myTI_2"/>
    <mx:Text id="myText_2"/>

    <mx:Script>
        <![CDATA[
            import mx.binding.utils.BindingUtils;

            // Define data binding in AS3
            public function initBindingHandler():void {
                BindingUtils.bindProperty(
                    myText_2, "text", //target
                    myTI_2, "text"); //source
            }
        ]]>
    </mx:Script>
</mx:Application>
```

Data Binding: Beispiel mit Custom Property

```
/* The whole class (all Getters/Setters) is bindable */  
[Bindable] public class MyCustomClass{  
    private var _description:String;  
  
    public function set description(value:String):void{  
        _description = value;}  
  
    public function get description():String{  
        return _description;}  
}
```

```
<!-- bind to property 'description' of custom class -->  
<mx:TextInput id="input" />  
<mx:Button label="set text" click="{myClass.description = input.text}"/>  
<mx:Text id="output" text="{myClass.description}"/>  
  
<mx:Script> <![CDATA[  
    import customClasses.MyCustomClass;  
    [Bindable] private var myClass:MyCustomClass = new MyCustomClass();  
]]></mx:Script>
```


Übung 1:

Data Provider

Übung 1

Aufgabe: Erweitert die AS3 Klasse Playlist, indem ihr

- ein List Objekt namens `_liste` einfügt
- zwei globale Variablen `_listData` (ArrayCollection) und `_selectedURL` (String) hinzufügt und jeweils Getter und Setter für sie schreibt
- der Setter für `_listData` setzt den Data Provider von `_liste`

Beim Auftreten des `creationComplete` Events der Anwendung, soll diese `_listData` der Playlist mit Demodaten füllen.

Bei Klick auf einen Playlisteintrag soll `_selectedURL` in Playlist entsprechend gesetzt werden. Bindet die `_url` des `MediaPlayers` an `_selectedURL` der Playlist.

XML Parsing

- Flex Framework beinhaltet die Klasse `XML`. Diese kann:
 - XML parsen
 - XML Strukturen erzeugen und verändern
- relative Navigation im XML Objekt:
`xml.descendants("Knotenname");`
`xml.attributes();`
- absolute Navigation im XML Objekt:
`xml.Knotenname.Knotenname.usw;`
`xml.@Attributname;`
- Beim Zugriff auf die Knoten des XML Objektes wird eine `XMLList` zurückgegeben. Dabei handelt es sich um ein Array von XML Objekten.

XML Parsing: Beispiel

```
<news>
  <item id="0">
    <title>Krass: Rollstuhlrennen im Altersheim</title>
    <description>Geschwindigkeit mal ganz langsam </description>
  </item>
</news>
```

```
if (xmlData.hasOwnProperty("item"))
{
  for each (var item:XML in xmlData.item as XMLList)
  {
    output.htmlText += "\nNews item number "+item.@id + "\n";

    if (item.hasOwnProperty("title"))
      output.htmlText += "<b>"+item.title + "</b>\n";

    if (item.hasOwnProperty("description"))
      output.htmlText += item.description + "\n";
  }
}
```

XML laden

- Verwendung der Klassen URLLoader und URLRequest

```
private function loadXML():void
{
    // Create URLLoader
    var loader:URLLoader = new URLLoader();

    // add event listener
    loader.addEventListener(Event.COMPLETE, onXMLLoaded);
    loader.addEventListener(IOErrorEvent.IO_ERROR, onXMLError);

    // create URLRequest and load the XML file
    var request:URLRequest = new URLRequest("assets/xml/test.xml");
    loader.load(request);
}
```

XML Namespaces

- enthält die XML-Datei einen oder mehrere Namensräume, müssen diese vor dem Parsen der entsprechenden Elemente gesetzt werden

```
default xml namespace = new Namespace("Namensraum") ;
```

Web Services

- Unterstützung für:
 - WSDL 1.1 oder höher
 - RPC-encoded
 - RPC-literal
 - document-literal
 - SOAP 1.1 oder höher
 - XML Schema 1.0
- Hinweis: Sandbox-Einschränkungen beachten! Web Service muss in gleicher Domäne liegen oder crossdomain.xml bereitstellen.

Web Services: Beispiel

```
<mx:WebService id="WS"
    wsdl="http://ws.cdyne.com/WeatherWS/Weather.asmx?wsdl"
    fault="Alert.show(event.fault.faultString), 'Error'">

    <mx:operation name="GetCityWeatherByZIP" resultFormat="xml">
        <mx:request>
            <ZIP>10001</ZIP>
        </mx:request>
    </mx:operation>
</mx:WebService>

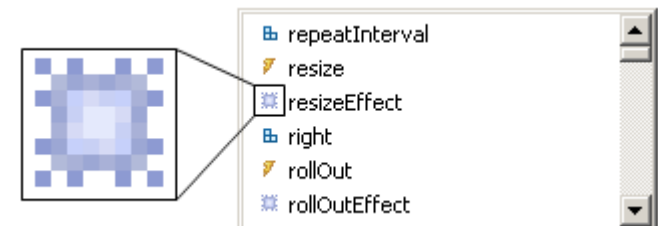
<mx:Button
    label="get New York weather"
    click="WS.GetCityWeatherByZIP.send()"
/>
<mx:Text
    htmlText="{WS.GetCityWeatherByZIP.lastResult}"
    width="100%"
/>
```


Effekte & Animationen

- erlauben es Animation und Bewegung zu GUI Objekten hinzuzufügen
- sind miteinander kombinierbar
- `mx.effects` enthält alle vorgefertigten Effekttypen, z.B:
 - Fade
 - Move
 - Rotate
 - Zoom
 - Parallel / Sequence
 - AnimateProperty

Effekte & Animationen

- können so zu einem Objekt hinzugefügt werden, das dieses Objekt automatisch beim Eintreten eines bestimmten Events den Effekt startet (z.B. `resizeEffect`, `creationCompleteEffect`)
- können auch manuell ausgelöst werden
- Online Doku: listet die automatisch auslösbbaren Effekte auf, die für ein GUI Objekt definiert werden können
- FlexBuilder: in der Liste der Code Completion werden diese Effekte für ein Objekt so dargestellt:



Effekte: Beispiel für automatisches Starten eines Effekts

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <!-- Add Resize Effect. The Effect is played automatically
    when the mouseDown event occurs on the Panel -->
    <mx:Panel id="myPanel"
        width="100"
        height="100"
        mouseDownEffect="resizeEffect"
    />

    <mx:Resize id="resizeEffect"
        widthFrom="100"
        widthTo="200"
        heightFrom="100"
        heightTo="200"
        duration="500"
    />

</mx:Application>
```

Effekte: Beispiel für manuelles Starten eines Effekts

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Panel id="myPanel"
        width="100"
        height="100"
    />

    <!-- Define effect and its duration in milliseconds -->
    <mx:WipeLeft id="myWL"
        duration="1000"
        target="{myPanel}"
    />

    <!-- Assign effect to target. Effect will play automatically
    when the mouseDown event occurs -->
    <mx:Button label="Play effect" click="{myWL.play()}" />

</mx:Application>
```

Übung 2: Effekte & Animationen

Übung 2

Aufgabe: Verseht die Komponenten der Anwendung mit folgenden Effekten:

- Klick auf Pause Button: ein Fade Effekt lässt das VideoDisplay innerhalb von 0,5 Sekunden halb transparent werden.
- Klick auf Start Button: Fade Effekt wieder rückgängig

Zusatzaufgabe:

MediaPlayer-Komponente und Playlist-Komponente:

- Cursor über TitleWindow: Fenster erhält Glow Effekt und der Rand des Fensters wird komplett undurchsichtig
- Cursor verlässt TitleWindow: vorherige Effekte rückgängig

Nativer 3D-Support

- verfügbar ab Flash Player 10
- ermöglichen einfache 3D-Effekte wie Rotation und Translation in einem 3D-Raum sowie Einstellung von Brennweite (focal length) und Bildausschnitt (field of view)
- verzerrt 2D-Objekte mittels Matritzen, so dass ein 3D-Eindruck entsteht
- kein Depth-Sorting implementiert; keine vorgefertigten 3D-Objekte verfügbar
- für komplexere 3D-Anwendungsfälle (3D-Modelle importieren, Shader und Lichtquellen setzen, Szenegraph, ...) sollten Third-Party-Frameworks wie z.B. Papervision3D verwendet werden

Nativer 3D-Support

- Translation: Manipulation der Attribute `x`, `y`, `z`
- Rotation: Manipulation der Attribute `rotationX`, `rotationY`, `rotationZ`
- Kameraperspektive: Manipulieren der Transformation mittels der Klasse `PerspectiveProjection`
- komplexere, benutzerdefinierte 3D-Effekte: `Matrix3D`, `Vector3D`, `Utils3D`, `Graphics.drawTriangles()`
- Beispiele
 - <http://www.yswfblog.com/blog/2008/11/10/3d-with-flash-10-simple-way-first/>
 - <http://www.yswfblog.com/blog/2008/11/12/3d-with-flash-10-part-2/>

Nativer 3D-Support: Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel
  xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="onCreationComplete()"
>
  <mx:Script><![CDATA[
    private function onCreationComplete():void
    {
      var pp:PerspectiveProjection = new
                                   PerspectiveProjection();

      pp.fieldOfView = 20;
      pp.projectionCenter = new Point(0, 0);
      this.transform.perspectiveProjection = pp;

      this.z = 100;
      this.rotationY = 20;
    }
  ]]></mx:Script>
</mx:HBox>
```

Übung 3: Nativer 3D-Support

Übung 3

Aufgabe:

Fügt der der Applikation in einer `init()`-Methode des `ControlPanel`s eine neue `PerspectiveProjection` hinzu.

Fügt dem `ControlPanel` folgende Funktionalität hinzu:

- Field of View Slider: verändert `fieldOfView` der Projektion
- Projektionsmittelpunkt X und Y Slider: verändern den Mittelpunkt der Projektion
- X-, Y- und Z-Rotaionswinkel Slider: Verändern X-, Y- und Z-Rotation von `MediaPlayer` und `Playlist`
- Z-Position Slider: verändern Z-Position von `MediaPlayer` und `Playlist`

Übung 3

Aufgabe (Fortsetzung):

Wenn sich die URL des abzuspielenden Videos im Mediaplayer ändert, soll dieser sich um 360 Grad um die X-Achse drehen.

Zusatzaufgabe:

Fügt dem Tab „Rotationseffekt“ im ControlPanel folgende Funktionalität hinzu:

- NumericStepper „Dauer“: stellt die Dauer des Effekts ein
- Radiobuttons „Achse“: setzen die Drehachse des Effekts
- Checkbox: aktiviert/deaktiviert den Effekt

Evaluation & Zertifikat

Zusammenfassung & Feedback