Uwe Mutz

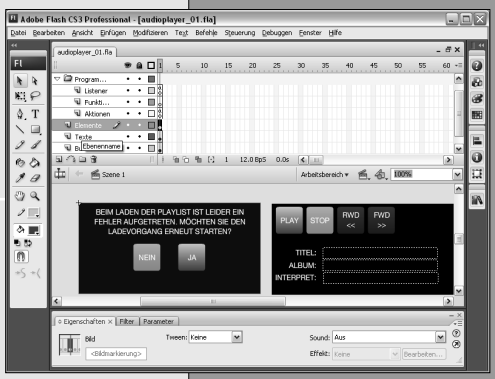
Flash CS3, AJAX & PHP

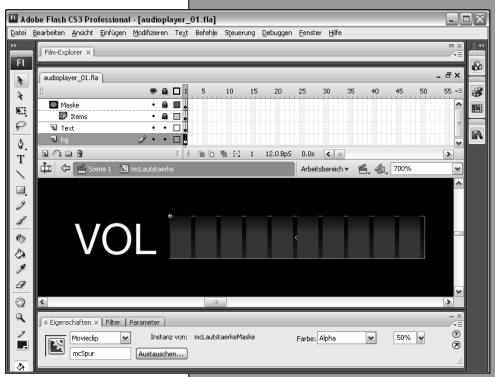
Dynamische, datenbank-basierte multimediale Anwendungen entwickeln

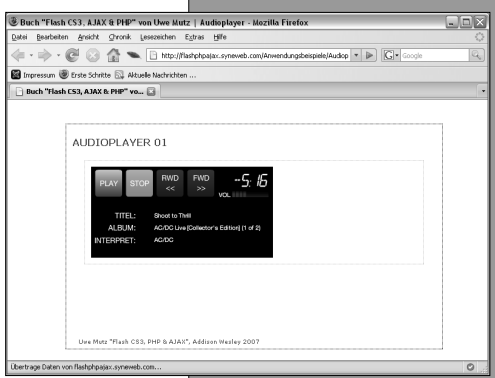
ADDISON-WESLEY

7

Audio-Jukebox







Dieses und die folgenden Kapitel widmen sich ausschließlich der Praxis. Hier sollen Sie sehen, auf welche Arten wir uns ein Zusammenspiel von Flash und PHP vorstellen.

Anwendungsbeispiele gibt es wohl wie Sand am Meer, Ideen und Vorstellungen zu Anwendungen noch viele mehr. Aus diesem Grund werden Sie auch zu diesem Buch gegriffen haben. Die serverseitige Programmierung als (hauptsächlicher) Datenlieferant auf der einen Seite, Flash als das Animations- und Multimedia-Tool auf der Clientseite.

Zu den gängigsten Aufgabengebieten finden Sie im Folgenden konkret durchgeplante und durchprogrammierte Übungen. Ich habe in allen Workshops die Flash- und die PHP-Entwicklung strikt getrennt, sodass Sie ein Optimum an Übersicht genießen können.

Die Audio-Jukebox wird zunächst in ActionScript 2.0 und später in ActionScript 3.0 entwickelt.

Viel Spaß beim Umsetzen und Anwenden!

7.1 Abfragen serverseitiger Informationen

Eine Variante, sich PHP für Flash zunutze zu machen, liegt in der simplen Abfrage von Daten aus einer Datenbank oder einem Filesystem. Dieses Kapitel widmet sich genau dieser Thematik. Als konkrete Anwendungen habe ich an dieser Stelle bevorzugt an multimediale Anwendungen gedacht, da sie im Allgemeinen keine Rückgaben an den Server erfordern.

7.2 Multimediale Anwendungen

Was ist Multimedia? Multimedia ist das Verwenden verschiedener Ausgabemedien zur selben Zeit. Hierzu zählt man Inhalte wie Text, Audio, Video usw. Je mehr dieser Assets zur gleichen Zeit ausgegeben werden, desto interessanter gestaltet sich die Entwicklung in Flash. Einmal ganz ehrlich – kennen Sie ein Tool, das besser für Multimedia auf Clientseite geeignet ist als Flash? Wohl kaum.

7.3 Konzept der Jukebox

Als erstes Beispiel dient dabei mein persönlicher Favorit. Kaum eine andere Anwendung hat mir in der Verwendung wie in der Entwicklung mehr Freude bereitet.

Am Anfang steht das Entwickeln eines Konzepts, also mehr oder weniger die Antwort auf die Frage: „Was soll das Tool eigentlich können?“ Hier die Anforderungen an unsere Audio-Jukebox aus Flash-Sicht:

1. Die Audio-Jukebox soll in der Lage sein, gestreamt MP3-Dateien abzuspielen. Somit ist die erste Anforderung, dass MP3-Dateien dynamisch in Flash eingelesen werden sollen.
2. Innerhalb der Jukebox soll volle Kontrolle über das Abspielen der Songs gewährleistet sein. Dies schließt Buttons für: „Start“, „Stopp“, „Ein Song weiter“ und „Ein Song zurück“ ein. Des Weiteren muss die Lautstärke regelbar sein.
3. Die gängigen Informationen zu den Songs sollen direkt aus den ID3-Tags der MP3-Dateien ermittelt und ausgegeben werden.
4. Die Restabspielzeit zum jeweiligen Song soll ausgegeben werden.
5. Es soll eine parametergesteuerte Autostart-Möglichkeit vorhanden sein.

ID3-Tags

Die ID3-Tags sind (benannte) Informationen wie „Titel“, „Interpret“, „Album“ usw., die mit der MP3-Datei gespeichert werden können. Geeignete Programme – wie auch Flash – können diese Tags auslesen.

7.3.1 Playlist generieren

Grundsätzlich stellt sich natürlich die Frage, wie Flash zu den Songs kommt, also woher die Playlist kommt bzw. wer sie generiert und wie sie eingelesen wird. Unser

Plan soll sein, dass sämtliche in einem Verzeichnis aufgelisteten MP3-Dateien verwendet werden. Eine auf diese Art erzeugte Playlist soll per Aufruf von Flash eine XML-Datei generieren, die Flash wiederum einliest.

XML-Struktur

Aus Sicht von Flash ist es vollkommen irrelevant, „wer" (der Webdesigner in Form einer Textdatei, ein Server basierend auf PHP-Code usw.) und auf welche Art die Playlist als XML-Datei generiert wird – Hauptsache, die Struktur ist bekannt und die XML-Daten stehen zur Verfügung. Punkt eins ist also, dass die Struktur der XML-Datei feststehen muss.

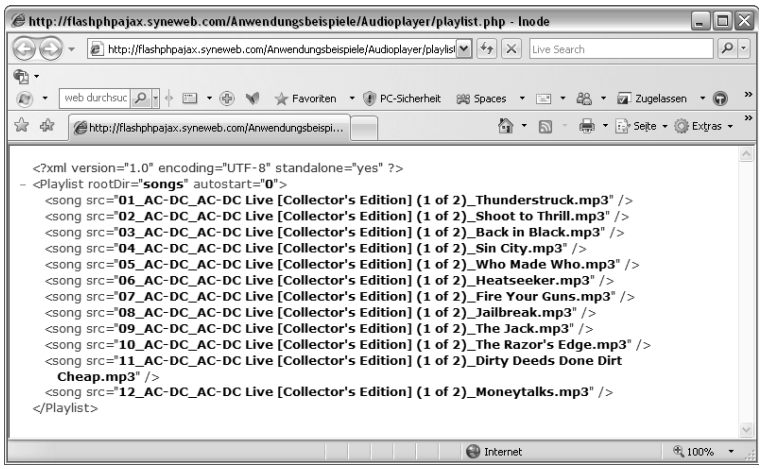


ABBILDUNG 7.1

Auf diese Art und Weise werden die MP3-Informationen an die Flash-Datei übergeben. XML is on its way.

Abbildung 7.1 verdeutlicht, wie die Informationen an die Flash-Anwendung übergeben werden sollen. Neben den Tags für die einzelnen Songs wird in zwei Attributen festgelegt, welches das Stammverzeichnis der Songs ist (`rootDir`) und ob ein Autostart – also ein automatisches Abspielen der Playlist – stattfinden soll (`autostart`). Wird die Variable `autostart` auf `0` gesetzt, so findet kein Autoplay statt, steht sie hingegen auf `1`, erfolgt das Abspielen der Playlist automatisch.

Flash-Datei vorbereiten

Gehen wir's an. Zunächst gliedern wir die Flash-Datei in drei gedachte Bereiche:

- Im ersten Bereich soll das Laden der XML-Daten erfolgen.
- Der zweite Bereich ist dafür gedacht, dem User eine Fehlermeldung auszugeben, sollte das Laden der XML-Daten nach einer gewissen Zahl von Wiederholungen nicht möglich gewesen sein.
- Der dritte Bereich stellt die eigentliche Jukebox dar. Dort befinden sich die Abspielsteuerung und die Informationsdarstellung der MP3-Daten.

7.4 Jukebox in ActionScript 2.0

Nachdem diese Vorkehrungen getroffen sind, konzentrieren wir uns nun wirklich auf das Programmieren. Zunächst muss definiert werden, aus welcher Datei die Playlist geladen werden soll – hierzu bedienen wir uns einer Stringvariablen namens `mySource`, die den Namen (oder auch eine komplette URL inklusive eventueller Unterverzeichnisse) der Playlist-Datei speichert. Des Weiteren legen wir – wie auch schon in den Theoriekapiteln gezeigt – in einer Variable `maxLadeversuche` eine gewisse Anzahl an Ladeversuchen fest, sollte das Laden der Daten nicht beim ersten Mal klappen. Hierzu muss in weiteren Variablen mitgezählt werden, wie viele Ladeversuche bereits vorgenommen wurden. Dies übernimmt die Variable `anzLadeversuche`.

```
var anzLadeversuche:Number = 0;
var maxLadeversuche:Number = 3;
var mySource:String = "playlist.php";
```

Listing 7.1: Die ersten drei wichtigen Variablen für die mitgezählte Anzahl an Ladeversuchen (`anzLadeversuche`), die maximale Anzahl an Ladeversuchen (`maxLadeversuche`) und die URL zur Playlist-Datei (`mySource`) sind somit angelegt.

Neben den oben angeführten Variablen werden wir im Weiteren noch einige zusätzliche globale Variablen benötigen, nämlich ein Array, das alle unsere eingelesenen Songs speichert (`theSongs`), zwei Variablen für die Ausgabe von Informationen zu den Songs in dynamischen Textfeldern (`theSongInfo1`, `theSongInfo2`) sowie eine Variable für die Darstellung der verbleibenden Abspielzeit, ebenfalls in einem dynamischen Textfeld (`theTimeInfo`).

```
var theSongs:Array = new Array();
var songTitel:String = "loading Playlist...";
var songInterpret:String = "";
var songAlbum:String = "";
var theTimeInfo:String = "--:--";
```

Listing 7.2: Anlegen einiger wesentlicher Variablen, die wir für den weiteren Ablauf der Jukebox benötigen werden

Natürlich müssen die Variablen `songTitel`, `songInterpret`, `songAlbum` und `theTimeInfo` nicht notwendigerweise genau zu diesem Zeitpunkt angelegt werden. Dies empfiehlt sich jedoch, da man (als guter Programmierer) davon ausgehen kann, dass man bis zum Abspielen der Songs kommt und somit diese Variablen auch benötigen wird.

7.4.1 XML-Daten laden

Nun kümmern wir uns darum, dass die XML-Daten geladen werden können. Das Flash-Objekt XML wird diese Aufgabe für uns erledigen. Obwohl wir keine leeren Textknoten zu erwarten haben, ist das Setzen der Eigenschaft `ignoreWhite` des XML-Objekts auf den Wert `true` eine gute Wahl, denn so kann diese mögliche Fehlerquelle ausgeschlossen werden.

```
var myLoader:XML = new XML();
myLoader.ignoreWhite = true;
myLoader.load(mySource);
```

Listing 7.3: Es wird ein neues XML-Objekt erzeugt, die Eigenschaft `ignoreWhite` *auf* `true` *gesetzt und von der zuvor definierten URL (*`mySource`*) werden XML-Daten eingelesen.*

`onLoad` **versus** `loaded`

Über das Ereignis `onLoad` des `XML`-Objekts wird signalisiert, dass das Laden der Daten abgeschlossen ist. Dies bedeutet aber noch nicht, dass das Laden auch **erfolgreich** war. Das erfolgreiche Laden signalisiert erst die Eigenschaft `loaded`. Dies bedeutet für uns, dass wir zunächst das Ereignis `onLoad` auswerten (wir erzeugen eine sogenannte Ereignisroutine oder Ereignisprozedur) und danach abprüfen müssen, ob die Eigenschaft `loaded` auf `true` gesetzt ist. Sollte dem nicht so sein, wird ein erneuter Ladeversuch gestartet. Das Definieren der Ereignisroutine muss kein zweites Mal vorgenommen werden, da beim erneuten Ladeversuch dasselbe `XML`-Objekt ein weiteres Mal verwendet wird. In jedem Fall wird die Variable `anzLadeversuche` um eins erhöht.

```
myLoader.onLoad = function():Void {
  anzLadeversuche++;
  if(myLoader.loaded) {
     //Hier folgt das Auslesen der Attribute und Tags.
     //Der Code wird weiter unten erläutert.
  }
  else {
     if(anzLadeversuche<maxLadeversuche) { myLoader.load(mySource); }
     else {
     //Code, falls Anzahl der Ladeversuche erreicht ist
     }
  }
}
```

Listing 7.4: Die Ereignisbehandlungsroutine wertet die Daten bei erfolgreichem Laden aus, bei nicht erfolgreichem Laden wird – sofern die Maximalanzahl an Ladeversuchen noch nicht erreicht ist (also `anzLadeversuche<maxLadeversuche`*) – ein erneuter Ladeversuch gestartet. Ansonsten müssen wir uns etwas einfallen lassen.*

Attribute speichern

War das Laden erfolgreich (und davon gehen wir einmal aus), werden zunächst alle im zweiten Tag (`<Playlist>`; siehe Abbildung 7.1) vorkommenden Attribute als globale Variablen gespeichert.

Eine `for..in`-Schleife empfiehlt sich hier: Alle im Objekt `attributes` vorkommenden Eigenschaften (hier: Variablen) werden ausgelesen und durch die Funktion `_global` in die Kollektion der globalen Variablen übernommen.

Danach werden alle folgenden Tags angesprochen und das Attribut `src` wird ausgelesen. In diesem befinden sich die Namen der MP3-Dateien. Vergleicht man den nachfolgenden Code mit der Abbildung der einzulesenden XML-Datei, so sieht man, dass das erste Child `<Playlist>` des Hauptknotens `<?xml>` eine gewisse Anzahl weiterer Child-Knoten umfasst (die Tags `<song>`). Wie viele es sind, lässt sich über die Eigenschaft `length` dieser `childNodes`-Sammlung herausfinden. Diese Information verpackt man in eine `for`-Schleife, liest Zug um Zug jedes Tag und dessen Attribut `src` aus und speichert die ermittelten Werte im globalen Array `theSongs`.

Ist dieser Schritt vollzogen, ist sämtliches Einlesen der XML-Daten abgeschlossen und es kann mit dem weiteren Gang der Jukebox fortgefahren werden.

```
//Dieses Code-Fragment wird an oben erwähnter Stelle eingefügt
for(itm in myLoader.firstChild.attributes) {
  _global[itm] = myLoader.firstChild.attributes[itm];
}
for(var i:Number=0; i<myLoader.firstChild.childNodes.length; i++) {
  theSongs[i] = myLoader.firstChild.childNodes[i].attributes.src;
}
```

Listing 7.5: Das Auswerten und Speichern der eingelesenen Daten erfolgt in diesem Codefragment. Sobald dieses Prozedere abgeschlossen ist, ist der wesentlichste Schritt getan und es kann mit dem Abspielen der Songs begonnen werden.

Für den Fall, dass der Ladevorgang nicht erfolgreich beendet werden konnte und auch die weiteren Ladeversuche zu keinem positiven Ergebnis geführt haben, wird ein MovieClip eingeblendet, der eine entsprechende Fehlermeldung auf den Bildschirm ausgibt. Ob Sie hier nun dem User die Meldung ausgeben, dass das Laden nicht geklappt hat, oder zusätzlich einen Button einfügen, der bei Klick einen neuerlichen Ladezyklus einleitet, hängt von Ihrem persönlichen Geschmack ab. Wir sind der Ansicht, dass sofern drei Ladeversuche fehlgeschlagen sind, ein weiterer Ladezyklus auch keinen Sinn mehr macht. Natürlich ist es aber Ihnen überlassen, ob Sie diese Möglichkeit in Betracht ziehen wollen oder nicht.

Ich habe mich für die Variante „Fehlermeldung & Buttons zum Neuladen“ entschieden, die wie folgt aussehen könnte:

ABBILDUNG 7.2

Der User erhält eine Fehlermeldung sowie die Möglichkeit, den Ladevorgang erneut einzuleiten.

Hierzu legt man einen MovieClip `mcError` an, der zwei Buttons (`btnNein`, `btnJa`) beinhaltet, und platziert ihn der Einfachheit halber neben der Bühne (quasi im unsichtbaren Bereich – siehe Abbildung 7.3). Um den MovieClip `mcError` gleich „am richtigen Platz" zu haben, sollte er eingeblendet werden müssen, fügt man folgenden Code in der Aktionenebene ein:

```
mcError._visible = false;
mcError._x = 0;
mcError._y = 0;
```

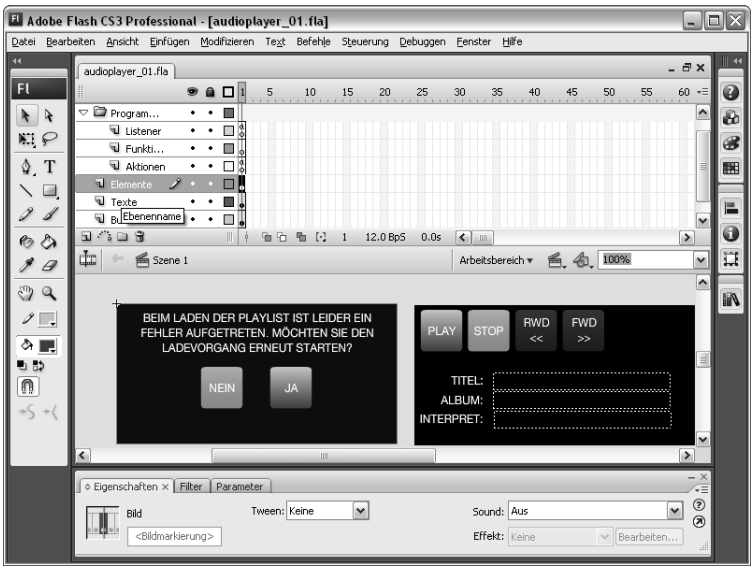


ABBILDUNG 7.3

Der MovieClip mcError wird im unsichtbaren Bereich neben der Bühne platziert.

Sollte die Anzahl der Ladeversuche erreicht sein (siehe Listing 7.4), fügt man folgenden Code ein:

```
...
if(anzLadeversuche<maxLadeversuche) { myLoader.load(mySource); }
else { mcError._visible = true; }
...
```

Listing 7.6: Die Zeile `mcError._visible = true;` *wird dem Code aus Listing 7.4 hinzugefügt.*

Nun verschaffen wir den beiden Buttons noch etwas „Intelligenz", indem wir Ihnen den Code zuweisen, den sie auslösen sollen:

```
mcError.btnNein.onRelease = function():Void {
  _root.mcError._visible = false;
  _root.songTitel = "ERROR WHILE LOADING PLAYLIST";
}
mcError.btnJa.onRelease = function():Void {
  _root.myLoader.load(mySource);
  _root.mcError._visible = false;
}
```

Listing 7.7: Die Eventhandler der Buttons btnNein und btnJa

Wird der Nein-Button geklickt, blenden wir den MovieClip `mcError` aus und geben in den Textfeldern eine Fehlermeldung aus, im Fall des Ja-Buttons wird zusätzlich noch der Ladevorgang erneut gestartet.

7.4.2 Abspielen der Songs

Befassen wir uns also nun mit dem wichtigen Thema des Abspielens der Songs. An dieser Stelle ist wieder etwas Konzept vonnöten. Überlegen wir uns, wie das Anwenden der Jukebox vonstatten gehen soll:

1. Über die globale Variable `autostart` wird entschieden, ob der erste Song abgespielt oder auf das Klicken des Play-Buttons durch den User gewartet werden soll.
2. Mithilfe der Buttons „Start", „Stopp", „Ein Song vor", „Ein Song zurück" soll man die Playlist abspielen und stoppen können. Des Weiteren wären kleine Icons (eines für jeden Song) nett, die bei Klick den Song gleich direkt aufrufen und abspielen.
3. Ein Lautstärkeregler muss ebenso vorhanden sein wie zwei Informationsfelder, eines zum abgespielten Song und eines zum Darstellen der verbleibenden Abspielzeit.

So weit der Plan. Programmiertechnisch ergeben sich hieraus vorerst einige wesentliche Konsequenzen:

1. Klarerweise wird ein Soundobjekt benötigt, um überhaupt Songs abspielen zu können.

2. In jedem Fall müssen wir uns merken, der wievielte Song aus unserer Songliste des Arrays `theSongs` zu einem gewissen Zeitpunkt abgespielt wird. Hierfür werden wir eine Variable `actualSong` verwenden, die auf den Index des Arrays `theSongs` zeigt.

Variablen für die Jukebox

Die beiden Punkte werden uns die wenigsten Kopfschmerzen bereiten, sind sie doch mit wenigen Zeilen erledigt.

```
var mySong:Sound = new Sound();
var actualSong:Number = 0;
```

Listing 7.8: Anlegen der beiden Variablen `mySong` *(Soundobjekt) und* `actualSong` *im Bild* `LadevorgangErfolgreich` *in der Ebene „Aktionen"*

Autostart der Songs

Widmen wir uns wieder den einfacheren Aufgaben, nämlich dem Auswerten der `autostart`-Variable. Wie schon weiter oben angesprochen, wird im Fall von `autostart==1` der erste Song (entspricht dem Wert von `actualSong`) abgespielt, indem die Funktion `playSong` aufgerufen wird. Auf diese Funktion wird weiter unten eingegangen. Ansonsten wird im oberen der beiden dynamischen Textfelder der Text „press the PLAY-Button to play songs …" ausgegeben. Das zweite Textfeld bleibt nach wie vor leer.

Um diesen Vorgang in Gang zu setzen, muss zunächst in Listing 7.5 nach dem Auslesen der XML-Daten der Aufruf der Funktion `initPlayer` eingefügt werden:

```
initPlayer();
```

Diese Funktion erledigt dann die oben genannten Aufgaben:

```
function initPlayer():Void {
  if(parseInt(autostart)==1) { playSong(0); }
  else {
     theSongInfo1 = "(press the PLAY-Button to play songs...)";
  }
}
```

Listing 7.9: Die `if`*-Bedingung entscheidet über* `autostart==1` *darüber, ob der erste Song abgespielt wird oder ob bei* `autostart!=1` *gewartet und eine entsprechende Meldung übergeben werden soll.*

Hauptfunktion der Jukebox: playSong()

Wenden wir uns nun der Funktion `playSong` zu, die einen wesentlichen Bestandteil der Jukebox darstellt. Wie bereits im letzten Listing gezeigt, wird an diese Funktion ein noch nicht näher beschriebener Parameter übergeben. Dieser Parameter wird verwendet, um genau um den Wert dieses Parameters in der Liste der Songs weiterzuspringen. Dabei kann dieser Parameter drei Werte annehmen, wie Sie der nachfolgenden Tabelle entnehmen können:

Wert	Bedeutung
`0`	Zum aktuellen Songindex (`actualSong`) wird der Wert 0 hinzugezählt und dieser Song abgespielt. Dies bedeutet also nichts anderes, als dass der aktuelle Song erneut abgespielt wird.
`1`	Zum aktuellen Songindex (`actualSong`) wird der Wert 1 hinzugezählt und dieser Song abgespielt. Es wird demnach in der Liste der Songs (`theSongs`) um einen Song weitergesprungen. Was geschehen soll, wenn das Ende der Liste erreicht ist, muss noch definiert werden.
`-1`	Zum aktuellen Songindex (`actualSong`) wird der Wert -1 hinzugezählt und dieser Song abgespielt. Es wird demnach in der Liste der Songs (`theSongs`) um einen Song zurückgesprungen. Was geschehen soll, wenn der Anfang der Liste erreicht ist, muss noch definiert werden.

Tabelle 7.1: Mögliche Übergabewerteparameter der `playSong`*-Funktion*

Anfang und Ende der Playlist

Innerhalb der Funktion `playSong` wird der Parameter als numerische Variable mit dem Namen `dir` (für „direction" – Richtung) geführt. Was soll nun geschehen, wenn entweder das Ende oder der Anfang der Songliste überschritten wird? Wir haben uns entschieden, dass bei Überschreiten des Listenendes an den Anfang der Liste und bei Überschreiten des Listenanfangs an das Ende der Liste gesprungen werden soll.

Das Überschreiten des Listenendes erkennt man daran, dass der Listenindex von `actualSong` gleich der Länge der Liste ist – die Zählung der Listeneinträge im Array beginnt ja bekanntlich bei 0. Das Überschreiten des Listenanfangs erkennt man wiederum daran, dass der Index den Wert -1 hat: `actualSong==-1`.

Songindex auslesen

Hat die Variable `actualSong` schlussendlich den gewünschten Wert erhalten, kann mit dem Laden des entsprechenden Songs aus der Liste begonnen werden. Man liest hierfür aus der Playlist `theSongs` den Wert aus, der über den Index `actualSong` definiert ist: `theSongs[actualSong]`. Vor diesen Eintrag hängt man noch das Stammverzeichnis der Songs und einen Slash „`/`". Das Stammverzeichnis wird über die globale Variable `rootDir` definiert, die aus der XML-Datei eingelesen wird, und ohne schließendes „/" angegeben.

Fertig ist die URL, von der Flash die Songdatei beziehen soll. Die Methode `loadSound` des Soundobjekts erwartet diese URL als ersten Parameter sowie eine Kennzeichnung, ob (`true`) oder ob die Datei nicht (`false`) gestreamt geladen werden soll, als zweiten Parameter:

```
mySong.loadSound("URL", isStreaming)
```

Das nachfolgende Listing zeigt Ihnen einen Auszug der Funktion `playSong`, worin die soeben dargestellten Aufgaben erledigt werden. Das vollständige Listing wird am Ende dieses Abschnitts aufgeführt.

```
function playSong(dir:Number) {
  ...
  actualSong += dir;
  if(actualSong==theSongs.length) { actualSong = 0; }
  if(actualSong==-1) { actualSong = theSongs.length-1; }
  mySong.loadSound(rootDir+"/"+theSongs[actualSong],true);
  ...
}
```

Listing 7.10: Codefragment zum Setzen der Variable `actualSong` *und Laden des entsprechenden Songs aus der Playlist* `theSongs`

Zwischen zwei Songs

Bevor jedoch die MP3-Datei geladen wird, sind noch ein paar „administrative" Punkte zu regeln:

1. Zwischenzeitlich (bis der neue Song gestartet ist) soll keine Restabspielzeit angezeigt werden, sondern etwas in der Art „--:--":
   ```
   theTimeInfo="--:--";
   ```
2. Gleiches gilt auch für die zwei Informationszeilen am Display:
   ```
   songTitel = "getting MP3-Information...";
   songInterpret = "";
   songAlbum = "";
   ```

Somit erweitert sich die Funktion `playSong` um folgenden Code:

```
function playSong(dir:Number):Void {
  theTimeInfo = "--:--";
  songTitel = "getting MP3-Information...";
  songInterpret = "";
  songAlbum = "";
  ...
}
```

Listing 7.11: Codefragment zum Erledigen der „administrativen" Angelegenheiten.

Das vollständige Script zur Funktion `playSong` sieht somit wie folgt aus:

```
function playSong(dir:Number):Void {
  theTimeInfo = "--:--";
  songTitel = "getting MP3-Information...";
  songAlbum = "";
  songInterpret = "";
  actualSong += dir;
  if(actualSong==theSongs.length) { actualSong = 0; }
  if(actualSong==-1) { actualSong = theSongs.length-1; }
  mySong.loadSound(rootDir+"/"+theSongs[actualSong],true);
}
```

Listing 7.12: Die Funktion `playSong` *in all ihrer Pracht!*

Song beendet: nächsten Song abspielen

Nun müssen wir uns noch überlegen, was zu geschehen hat, wenn der nun abgespielte Song zu Ende ist. Wir haben uns entschieden, nach einem fertig abgespielten Song ganz einfach den nächsten Song in der Liste abzuspielen. Die Methode `onSoundComplete` des Soundobjekts wird genau dann aktiv, wenn ein Song komplett wiedergegeben wurde. Nichts ist leichter, als zu diesem Zeitpunkt den nächsten Song in der Liste abzuspielen, denn die dafür erforderliche Funktion haben wir bereits: `playSong(1)`. Der Wert 1 als Übergabeparameter teilt der Funktion mit, dass das nächste Lied der Liste abgespielt werden soll.

```
mySong.onSoundComplete = function() {
  playSong(1);
}
```

Listing 7.13: Codefragment für die Ereignisbehandlung von `onSoundComplete`*, die direkt nach dem Anlegen der* `Sound`*-Instanz* `mySong` *definiert werden kann.*

Songs per Klick abspielen

Neben dem Abspielen des vorigen, aktuellen oder nächsten Songs soll eventuell auch noch die Möglichkeit bestehen, einen Song direkt anzuwählen. Dafür ist unsere Funktion `playSong` jedoch nicht ausgelegt – hierzu müsste man eine eigene Funktion programmieren, wo etwa die Songs in einer ComboBox aufgelistet werden. Kein Problem für uns – öffnen Sie die Datei *audioplayer_03.fla* von der Buch-CD – die Erklärung finden Sie weiter unten im Kapitel, da diese Variante in ActionScript 3.0 umgesetzt wurde.

7.4.3 ID3-Tags auslesen

Abschließend sollte man bei Auswahl eines neuen Songs natürlich auch die entsprechenden Informationen zu ihm erhalten. Wurde beim Aufnehmen der MP3-Daten darauf geachtet, dass die Informationen zum Album mitgespeichert werden, so werden diese Informationen beim Aufnehmen mit in die MP3-Datei „verpackt" und sind somit zu jeder Zeit verfügbar. Im Allgemeinen geschieht dies entweder durch automatisiertes Abfragen einer Datenbank (beispielsweise CDDB) durch Ihr Aufnahmeprogramm oder durch händisches Eingeben der Informationen vor der Aufnahme.

Da alle gängigen Aufnahmeprogramme wie zum Beispiel iTunes von Apple oder die Jukebox von MusicMatch (Yahoo) nach diesem Verfahren arbeiten, können wir Entwickler davon ausgehen, dass auch unsere MP3-Daten diese Informationen gespeichert haben. Flash kann – wie sollte es auch anders sein – auf diese Informationen zugreifen. Hierzu existiert für ein Soundobjekt das Sub-Objekt `id3`, dessen Eigenschaften genau die gewünschten Daten beinhaltet. Man nennt diese MP3-Informationen auch die „ID3-Tags", deshalb auch der Objektname `id3`. Die für uns interessanten Eigenschaften sind in der folgenden Tabelle zusammengefasst. Eine komplette Auflistung aller Eigenschaften finden Sie in der ActionScript-Referenz, die Sie von der Macromedia-Site downloaden können: `http://www.macromedia.com/support/documentation/de/flash/`. Alternativ dazu könnte man natürlich die notwendigen Informationen zum Song in die XML-Datei aufnehmen, da man – möchte man absolut sichergehen – nicht zu 100% davon ausgehen kann, dass die ID3-Tags wirklich in der MP3-Datei gespeichert sind.

Tag	Enthaltene Information
`id3.artist`	Name des Interpreten
`id3.songname`	Songtitel
`id3.album`	Name des Albums
`id3.TLEN`	Länge des Songs (in Millisekunden)

Tabelle 7.2: Auflistung der für unsere Jukebox interessanten Eigenschaften des `id3`*-Objekts der Version 1*

Neben den hier dargestellten `id3`-Tags existieren noch weitere, die einer neueren Generation angehören. Sie sind in Flash unter der Bezeichnung id3 Version 2 zu finden – genauere Informationen hierzu entnehmen Sie bitte der *Flash-Referenz*.

In der ersten Zeile des Displays sollen die Informationen zum Titel in Zeile 2 und in Zeile 3 die Infos zu Interpret und dem Albumtitel stehen. Wie diese Informationen in die dynamischen Textfelder gelangen, wird hier nicht gesondert erläutert. Sollten Sie dazu Fragen haben, schauen Sie sich den Code am besten direkt in der fla-Datei von der Buch-CD an.

Da die ID3-Informationen nicht sofort zur Verfügung stehen, sondern erst dann, wenn ein gewisser Teil der MP3-Datei geladen ist, müssen wir ein Ereignis abwarten, das uns das Vorhandensein dieser Informationen anzeigt: `onID3`.

```
mySong.onID3 = function():Void {
  songInterpret = mySong.id3.artist;
  songAlbum = mySong.id3.album;
  songTitel = mySong.id3.songname;
  showTimer(Math.floor(mySong.id3.TLEN/1000));
}
```

Listing 7.14: Codefragment zum Einlesen und Ausgeben der ID3-Informationen

Restspielzeit anzeigen

Die Darstellung der Restspielzeit gestaltet sich dabei auch etwas aufwändiger – hier werden wir eine eigene Funktion aufrufen, in der wir mit einem Intervall arbeiten. Dieser Funktion (`showTimer`) übergeben wir primär die Länge des Songs in Sekunden, weshalb eine Division durch 1000 und eine Rundung des Ergebnisses erforderlich ist. Alles weitere passiert dann innerhalb dieser Funktion.

Idealerweise realisiert man dies mithilfe der Methode `setInterval`. Wie in der Funktion `playSong` ersichtlich, wird in der Ereignisbehandlungsroutine des Ereignisses `onID3` die Funktion `showTimer` wie folgt aufgerufen:

```
showTimer(Math.floor(mySong.id3.TLEN/1000));
```

Der Übergabewert an `showTimer` ist die (abgerundete) Länge des MP3-File und wird innerhalb der Funktion in der Variable `theLength` gespeichert. Sobald die Funktion `showTimer` aufgerufen wird, wird zunächst ein (bis dato noch nicht verwendetes) Intervall gelöscht: `clearInterval(itvTimer)`. Der Grund dafür ist, dass im Weiteren über `showTimer` jede Sekunde dasselbe Intervall erneut gestartet wird, das wiederum die Funktion `showTimer` mit einem genau um eine Sekunde verminderten Wert für den Parameter aufruft.

Letzten Endes wird hier nicht die Funktionsweise des Intervalls, sondern vielmehr die eines Timers verwendet. Sollte `theLength>=0` sein – also das Lied noch nicht zu Ende sein –, wird der Wert für die Restabspielzeit wie folgt berechnet:

1. Zunächst werden die Minuten ermittelt: Hierzu dividiert man die Restzeit in Sekunden durch 60 und rundet ab:

   ```
   theTimeInfo = "-"+Math.floor(theLength/60)+":";
   ```

   Vor diesen Wert wird noch ein „–“ (Minus) gesetzt, um die Restdauer anzudeuten. Hinter den Minutenwert wird ein Doppelpunkt gesetzt.

2. Danach werden die restlichen Sekunden ermittelt, indem man den Minutenwert mit 60 multipliziert und diesen Wert von den Gesamtsekunden (`theLength`) abzieht:

   ```
   theLength-60*Math.floor(theLength/60)
   ```

   Sollte dieser Wert kleiner als zehn sein, so sollte man der Sekundenanzeige eine Null vorsetzen („09“ wirkt besser als „9“):

   ```
   if(theLength-60*Math.floor(theLength/60)<10) { theTimeInfo += "0"; }
   ```

   Der Sekundenwert wird schlussendlich auch noch an den beinahe schon kompletten String für die Ausgabe angehängt:

   ```
   theTimeInfo += theLength-60*Math.floor(theLength/60);
   ```

3. Nur zur Erinnerung: Die globale Variable `theTimeInfo` ist diejenige Variable, die im dynamischen Textfeld für die Restabspielzeit ausgegeben wird. Abschließend muss das Intervall nur noch erneut aufgerufen werden:

   ```
   itvTimer = setInterval(showTimer,1000,theLength-1);
   ```

Das vollständige Listing der Funktion `showTimer` sieht wie folgt aus:

```
function showTimer(theLength:Number):Void {
  clearInterval(itvTimer);
  if(theLength>=0) {
      theTimeInfo = "-"+Math.floor(theLength/60)+":";
      if(theLength-60*Math.floor(theLength/60)<10) { theTimeInfo += "0"; }
      theTimeInfo += theLength-60*Math.floor(theLength/60);
      itvTimer = setInterval(showTimer,1000,theLength-1);
  }
}
```

Listing 7.15: Das vollständige Listing der Funktion `showTimer`

7.4.4 Abspielsteuerung

Erfreulicherweise haben wir alle aufwändigeren Codes hinter uns gebracht und können uns nun einer einfacheren Funktion widmen: der Funktion `stopSound`, die für das Stoppen eines gerade abgespielten Songs verantwortlich ist. Dies geschieht über die einfache Methode `stop()` des Soundobjekts: `mySong.stop();`. Gleichzeitig mit dem Stoppen des Songs müssen noch zwei weitere Aufgaben erledigt werden:

Das laufende Intervall muss gelöscht werden:

```
clearInterval(itvTimer);
```

Das Display muss aktualisiert werden:

```
theTimeInfo = "--:--";
theSongInfo1 = "(stopped)";
theSongInfo2 = "";
```

Erledigt. Abschließend noch das Listing:

```
function stopSong():Void {
  mySong.stop();
  clearInterval(itvTimer);
  theTimeInfo = "--:--";
  songTitel = "(stopped)";
  songInterpret = "";
  songAlbum = "";
}
```

Listing 7.16: Code der Funktion `stopSong`

Kommen wir zu den Buttons, also der Abspielsteuerung unserer Jukebox. Sie werden sehen, dass die Buttons mit Sicherheit die einfachste Aufgabe darstellen:

- Button „Play“:

```
btnPlay.onRelease = function():Void {
  playSong(0);
}
```

- Button „Stopp“:

```
btnStop.onRelease = function():Void {
  stopSong();
}
```

- Button „Ein Song zurück“:

```
btnRWD.onRelease = function():Void {
  playSong(-1);
}
```

- Button „Ein Song vor“:

```
btnFWD.onRelease = function:Void {
  playSong(1);
}
```

7.4.5 Lautstärkeregler

Sie denken, wir hätten damit das Ende erreicht? Beinahe. Bleibt einzig noch der Lautstärkeregler, der eigentlich ganz simpel zu programmieren ist. Aber zunächst einige Gedanken zur Funktionsweise:

1. Durch Klicken auf die Spur des Reglers soll die Lautstärke auf den entsprechenden Wert gesetzt werden.

2. Die Lautstärke soll mit der Maus per Drag&Drop gesetzt werden können, wobei sich während des Ziehens die Lautstärke kontinuierlich verändern soll.

Diese aufgelisteten Punkte werden uns zum Ende hin noch einmal ein wenig schwitzen lassen, aber dafür haben wir es dann geschafft – zumindest in Flash.

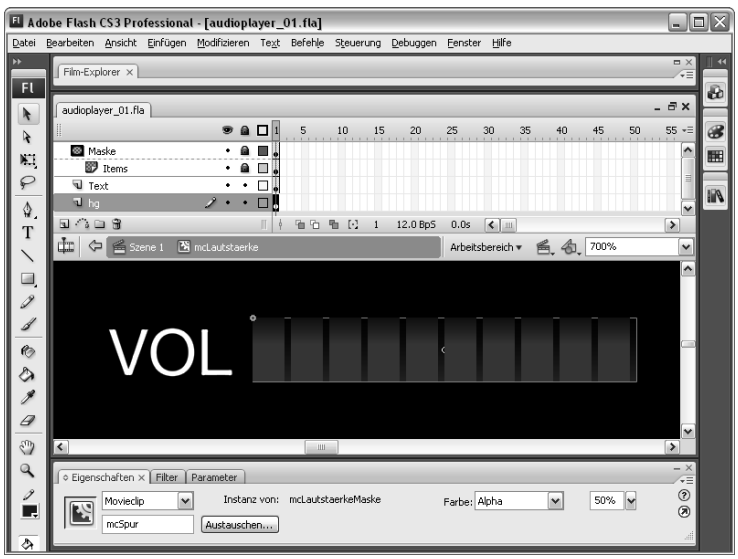


ABBILDUNG 7.4

Der Lautstärkeregler ist ein eigenes Symbol in der Bibliothek, ebenso wie die Spur & Maske des Reglers (MovieClip).

Das Symbol ist in vier Ebenen unterteilt:

- Ebene „hg“: Diese beinhaltet den MovieClip `mcSpur`, der zu 50% transparent ist. Er dient dazu, dass der User beim Verstellen der Lautstärke immer noch weiß, wie hoch er die Lautstärke stellen kann. Des Weiteren werden wir die Spur dafür verwenden, dass der User auf diese klicken und somit die Lautstärke verstellen kann.
- Ebene „Text“: Diese beinhaltet lediglich den Text „VOL“.
- Ebene „Items“: Sie beherbergt die zehn blauen Striche, die zur symbolhaften Darstellung der Lautstärke dienen.
- Ebene „Maske“: Sie legt fest, wie viel der Ebene „Items“ angezeigt wird. Als Maske wird derselbe MovieClip verwendet wie für die Spur (`mcLautstaerkeMaske` in der Bibliothek), da diese sowieso nie sichtbar sein wird.

In der Listener-Ebene auf der Hauptbühne platzieren wir den Code, der beim Drücken und Loslassen der Maustaste über dem Lautstärkeregler ausgeführt werden soll:

```
var itvLautstaerke;
mcLautstaerke.mcSpur.onPress = function():Void {
  itvLautstaerke = setInterval(setLautstaerke,50);
}
mcLautstaerke.mcSpur.onRelease = function():Void {
  clearInterval(itvLautstaerke);
}
mcLautstaerke.mcSpur.onReleaseOutside = function():Void {
  clearInterval(itvLautstaerke);
}
```

Listing 7.17: Der Code für die Lautstärkesteuerung ist relativ einfach, sehen wir einmal von der noch nicht definierten Funktion `setLautstaerke` *ab.*

Die Variable `itvLautstaerke` wird für ein Intervall verwendet, auf das wir später noch zu sprechen kommen. Interessanter wird es schon, wenn wir die Funktionenebene etwas näher betrachten:

```
function setLautstaerke():Void {
  var mousex:Number = mcLautstaerke._xmouse;
  var maxX:Number = mcLautstaerke.mcSpur._width;
  mcLautstaerke.mcMaske._width = mousex;
  mySong.setVolume(mousex/maxX*100);
}
```

Listing 7.18: In der Funktionenebene wird die Funktion `setLautstaerke` *definiert.*

Sobald die Funktion `setLautstaerke` aufgerufen wird, wird die Position der Maus innerhalb des Lautstärkereglers ermittelt (`mcLautstaerke._xmouse`). Da sich die Spur an der Position (0/0) befindet, entspricht die Mausposition in x-Richtung auch direkt der zu setzenden Lautstärke. Die Variable `maxX` fragt die maximale Lautstärke ab, indem sie die Breite der Spur ermittelt (`mcLautstaerke.mcSpur._width`). Danach wird die Breite der Maske für die Lautstärke-Items (siehe Erklärung zu Abbildung 7.4) noch auf genau den Wert gesetzt, wo die Maus des Users gerade steht. Abschließend errechnen wir aus Mausposition und maximaler Lautstärke die tatsächliche Lautstärke für den Song (in Prozent von 0–100) und weisen diese der Soundinstanz `mySong` zu (`mousex/maxX*100`).

Sprechen wir noch kurz über das Intervall `itvLautstaerke`. Das Problem mit einer Lautstärkeregelung in dieser Art und Weise ist, dass der Ablauf wie folgt sein soll:

1. Sobald der User auf den Lautstärkeregler klickt und die Maus noch gedrückt hält, soll die Lautstärke gesetzt werden – auch wenn der User dabei die Maus bewegt. Da das auftretende Ereignis das Ereignis `onPress` ist und dieses nur einmal auftritt (nämlich beim Hinunterdrücken der Maustaste), könnte die Lautstärke auch nur zu diesem Zeitpunkt gesetzt werden, unabhängig davon, wie lange er die Maustaste unten hält.

2. Beim Loslassen der Maustaste (innerhalb oder außerhalb des Reglers) soll das Setzen der Lautstärke wieder beendet werden. Die möglichen auftretenden Ereignisse nennen sich `onRelease` und `onReleaseOutside` – auch diese beiden treten nur einmal auf, was uns jedoch nicht behindert.

Der Ausweg aus der im ersten Punkt dargestellten Problematik ist, dass sobald der User die Maustaste nach unten drückt, ein Intervall `itvLautstaerke` startet, das in regelmäßigen Abständen (bei uns sind das 50 Millisekunden – entspricht einer Framerate von 20 Bildern pro Sekunde: 1/20 Sekunde pro Bild) die Lautstärke setzt (über den Aufruf der Funktion `setLautstaerke`). Beim Loslassen der Maustaste muss dieses Intervall nur noch wieder gelöscht werden.

7.5 Der PHP-Part

Nun geht's endlich mit der PHP-Programmierung los. Unser Ziel ist es, alle MP3-Dateien eines bestimmten Verzeichnisses unseres Webservers auszulesen und diese Daten in XML-Form an Flash zu übergeben. Sie werden sehen, dass dies auf einfache Art und Weise zu realisieren ist.

Zuerst müssen wir natürlich eine PHP-Datei erstellen. Der Name wird uns in diesem Fall von Flash vorgegeben, da beim Einlesen der XML-Datei die Datei *playlist.php* gewünscht wird. Somit nennen wir unsere Datei auch entsprechend „playlist.php".

Sofern Sie noch keinen Ordner mit MP3-Dateien auf Ihrem Server erstellt haben, sollten Sie das jetzt tun. Erstellen Sie den Ordner im gleichen Verzeichnis wie die Flash-Datei und nennen Sie ihn „songs". Zur Sicherheit sollten Sie bei der Benennung der Files auf Leer- und Sonderzeichen verzichten. So weit, so gut.

Um in unserer Programmierung flexibel zu bleiben, speichern wir den Ordner der MP3-Files und auch die Einstellung für den Autostart in jeweils einer eigenen Variable ab. Würde sich später der Ordnername ändern, müssten wir bloß den Wert der Variable korrigieren und nicht an jeder Stelle im Skript, an der auf den Ordner verwiesen wird.

```
if(!isset($_GET["dir"])) { $dir = "songs"; }
else { $dir = $_GET["dir"]; }
if(!isset($_GET["autostart"]) || is_nan($_GET["autostart"])) { $autostart = 0; }
else { $autostart = $_GET["autostart"]; }
```

Listing 7.19: Anlegen der Variablen für den MP3-Ordner (`$dir`) und den Wert von `$autostart`

Wie Sie sehen, kann per GET sowohl das auszulesende Verzeichnis der MP3-Files (`$_GET["dir"]`) als auch die Info über ein Autostart (`$_GET["autostart"]`) an die Datei übergeben werden, was unser Script sehr flexibel macht.

Verzeichnis auslesen

Um an die Dateien eines Verzeichnisses zu kommen und diese auszulesen, benötigen wir insgesamt drei PHP-Funktionen.

- `opendir()`: Mit der Funktion `opendir()` können wir den gewünschten Ordner öffnen. Diese Funktion erfordert nur den Pfad zum Ordner und logischerweise auch den Ordnernamen selbst.
- `readdir()`: Mit `readdir()` können wir immer eine Datei des geöffneten Ordners auslesen. Um alle Files nacheinander auszulesen, werden wir uns einer Schleife bedienen. Auch `readdir()` benötigt allein den Pfad zum Ordner und den Ordnernamen.
- `closdir()`: Diese Funktion ist notwendig, um das geöffnete Verzeichnis wieder zu schließen.

Da wir nun die notwendigen PHP-Funktionen kennen, ist es nicht mehr schwer, unsere MP3-Files auszulesen. Zuerst öffnen wir das benötigte Verzeichnis mit `opendir()`. Als Parameter geben wir einfach die Variable `$dir` aus Listing 7.19 an, da in dieser ja der Ordnername, in unserem Fall „songs“, gespeichert wurde.

```
$verz = opendir($dir);
```

Zur Erinnerung schauen wir uns noch einmal die gewünschte XML-Struktur an, die Flash benötigt.

```
<Playlist rootDir="Verzeichnis" autostart="0">
  <song src="MP3-Filename"></song>
  ...
</Playlist>
```

Listing 7.20: Die notwendige XML-Struktur, die es zu erzeugen gilt

Bei Betrachten des ersten Knotens sehen Sie, dass wir mit diesem bereits alle notwendigen Informationen angeben, also den Pfad zum Ordner, den Ordnernamen und den Wert für `Autostart`. Um später eine korrekte XML-Struktur mit der richtigen Formatierung erstellen zu können, speichern wir unsere Knoten in einer Variable `$playlist`. Diese Variable werden wir schlussendlich per `echo` in das Dokument ausgeben..

```
$playlist = ,<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>';
$playlist.= ,<Playlist rootDir="'.$dir.'" autostart="'.$autostart.'">';
```

Listing 7.21: Erstellen der XML-Definition des Dokuments sowie des ersten Knotens `<Playlist>`

Als Nächstes müssen wir die MP3-Files aus unserem Ordner auslesen. Dazu bedienen wir uns der PHP-Funktion `readdir()`, mit der man immer eine Datei eines bestimmten Ordners auslesen kann. Um alle Dateien des Ordners auslesen zu können, werden wir eine `while`-Schleife nutzen.

Die Funktion `readdir()` gibt den booleschen Wert `true` zurück, wenn das Lesen im Verzeichnis erfolgreich war. In unserer `while`-Schleife soll der Knoten `song` so lange erstellt werden, wie Dateien im Ordner „songs“ gefunden werden. Die gefundenen Dateien speichern wir in der Variable `$file` ab.

```
while($file = readdir($verz)) {
   ...
}
```

Daten prüfen

Ein guter Programmierer sollte immer alle notwendigen Daten überprüfen. In unserem Fall möchten wir nur Dateien vom Typ .mp3 in unsere XML-Struktur aufnehmen. Somit müssen wir zum einen kontrollieren, ob es sich generell um eine Datei (und keinen weiteren Ordner) handelt und ob diese vom Typ .mp3 ist. Dazu bietet uns PHP wieder Funktionen, die wir direkt nutzen können.

- `is_dir()`: Mit `is_dir()` prüfen wir, ob es sich um ein Verzeichnis handelt. In unserem Fall werden wir diese Abfrage negieren, da wir wissen wollen, ob es sich nicht um einen weiteren Ordner handelt.
- `substr()`: Diese Funktion gehört zu den Zeichenkettenfunktionen. `substr()` erwartet einen String und einen Startwert (Position im String), ab dem der Teilstring erzeugt werden soll. Als Rückgabe wird der restliche String ab dem definierten Startwert zurückgeben. Um die Zählung von hinten beginnen zu lassen, wird einfach der Startwert negativ gesetzt.

Somit ergibt sich folgender Code für die Abfrage:

```
if(!is_dir($file) && (substr($file,-3)=="mp3"))
```

Innerhalb der Abfrage erweitern wir die bereits vorhandene Variable `$playlist` um die eingelesene Datei:

```
$playlist .= ,<song src="'.$file.'" />';
```

Nachdem wir nun alle MP3-Files aus dem Ordner „songs“ gelesen haben, schließen wir außerhalb der `while`-Schleife den geöffneten Ordner wieder.

```
closedir($verz);
```

Abschließend wird das geöffnete `<Playlist>`-Tag wieder geschlossen, der Inhalt in eine UTF8-kodierte Form gebracht (sollte das nicht gewünscht sein, so entfernen Sie diese Zeile einfach aus Ihrem Script) und `$playlist` per echo ausgegeben:

```
$playlist .= ,</Playlist>';
$playlist = utf8_encode($playlist);
echo($playlist);
```

Das vollständige Skript sollte nun so aussehen:

```
if(!isset($_GET["dir"])) { $dir = "songs"; }
else { $dir = $_GET["dir"]; }
if(!isset($_GET["autostart"]) || is_nan($_GET["autostart"])) { $autostart = 0; }
else { $autostart = $_GET["autostart"]; }

$verz = opendir($dir);

$playlist = '<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>';
$playlist.= '<Playlist rootDir="'.$dir.'" autostart="'.$autostart.'">';
while($file = readdir($verz)) {
  if(!is_dir($file) && (substr($file,-3)=="mp3")) {
     $playlist .= '<song src="'.$file.'" />';
  }
}
closedir($verz);

$playlist .= ,</Playlist>';
$playlist = utf8_encode($playlist);
echo($playlist);
```

Listing 7.22: Erstellen der XML-Struktur durch Auslesen der MP3-Files aus einem gegebenen Ordner

Um die Jukebox zu testen, müsste die PHP-Datei in einer Serverumgebung (wie etwa dem XAMPP-Server oder online) getestet werden. Hierzu laden Sie die Datei inklusive der MP3-Dateien im angegebenen Ordner in das Verzeichnis, das Sie bei der Installation als Root-Verzeichnis angegeben haben, und rufen Sie die php-Datei im Browser auf:

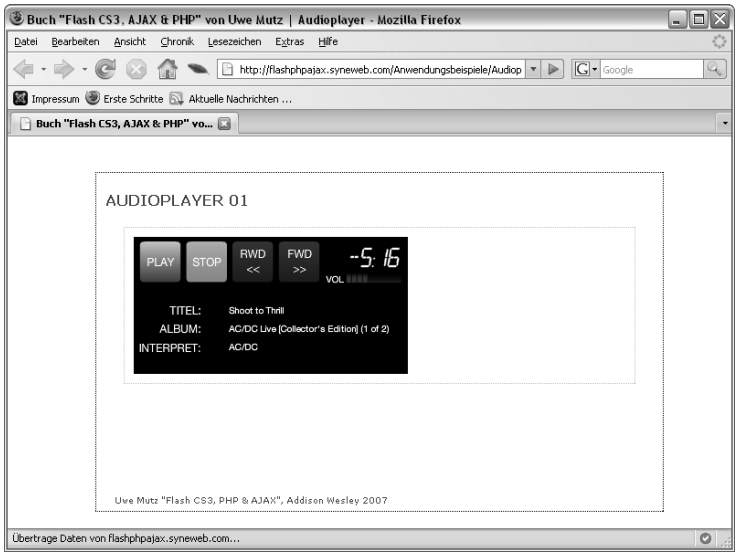


ABBILDUNG 7.5

Die Erfolgsbilanz ist positiv: Der Audioplayer läuft auch im Browser ohne Probleme!

7.6 Jukebox in ActionScript 3.0

In Kapitel 7.4 wurde die Jukebox in ActionScript 2.0 entwickelt – nun wenden wir uns ActionScript 3.0 zu und werden sehen, an welchen Stellen der Code (eventuell grundlegend) verändert werden muss. Idealerweise nehmen Sie sich die Datei *audioplayer_01.fla* von der Buch-CD zur Hand und verfolgen dort die Änderungen.

7.6.1 Allgemeine Änderungen

Ein wesentlicher Unterschied von ActionScript 2.0 auf ActionScript 3.0 ist die geänderte Schreibweise des reservierten Worts VOID:

- ActionScript 2.0: VOID wird mit einem großen Anfangsbuchstaben geschrieben: Void.
- ActionScript 3.0: VOID wird mit einem kleinen Anfangsbuchstaben geschrieben: void.

Dies betrifft alle unsere Funktionen ohne Rückgabewert und das sind 15 an der Zahl, die wir verteilt in den Ebenen Aktionen, Funktionen und Listener finden.

Weitere Änderungen sind:

- Sämtliche Variablen, die innerhalb von ActionScript 3.0 Verwendung finden, müssen explizit deklariert werden – eine implizite Deklarierung ist nicht möglich. Somit müssen die Attribute `autostart` und `rootDir`, die in der XML-Datei im Tag `<Playlist>` verwendet werden, explizit deklariert werden.

- Ein Zugriff auf globale Variablen ist nicht mehr möglich, da es keine globalen Variablen mehr gibt. Damit entfällt der Verweis auf `_global` komplett.
- Ebenso existiert der Zugriff auf `_root` nicht mehr – dieser lautet nun nur noch `root` (ohne Unterstrich am Anfang).
- Im Allgemeinen werden Eigenschaften eines MovieClips nun nicht mehr mit einem beginnenden Unterstrich „_“ angegeben – der Unterstrich entfällt komplett.
- Textfelder, die auf Variablennamen zugreifen, um so den Wert der Variable darzustellen, werden nicht mehr unterstützt – diese müssen nunmehr mit dem Feldnamen angesprochen werden.

7.6.2 Änderungen beim Laden der XML-Daten

Da in ActionScript 3.0 ein allgemeines Objekt `URLRequest` in Verbindung mit `URLLoader` zum Laden von Daten eingeführt wurde, läuft das Laden von XML-Daten nun auch anders ab – damit haben wir uns schon recht ausführlich im vorigen Kapitel befasst. Anstatt die `load`-Methode des XML-Objekts zu verwenden, wird nun mit `URLLoader` und `URLRequest` gearbeitet. Somit wird:

```
var myLoader:XML = new XML();
myLoader.ignoreWhite = true;
...
myLoader.load(mySource);
```

zu:

```
var myRequest:URLRequest = new URLRequest(mySource);
var myLoader:URLLoader = new URLLoader();
var myXML:XML;
...
myLoader.load(myRequest);
```

Die Eigenschaftszuweisung `ignoreWhite = true` darf erst dann passieren, wenn der XML-Instanz wirklich XML-Daten zugewiesen wurden.

Der Eventhandler, der beim vollständigen Laden der Daten aufgerufen wird, muss auch entsprechend abgeändert werden. Aus:

```
myLoader.onLoad = function():Void {
  ...
}
```

wird:

```
function EXMLComplete(myEvent:Event):void {
```

```
    ...
  }
  myLoader.addEventListener(Event.COMPLETE, EXMLComplete);
```

Innerhalb des Eventhandler `EXMLComplete` wird nun ebenfalls auf geänderte Art und Weise auf die XML-Daten zugegriffen. Anstatt die Elemente des XML-Baums Schritt für Schritt durchzuparsen, können wir direkt auf die Elemente zugreifen. Die gesamte Funktion erhält nun folgenden Inhalt:

```
function EXMLComplete(myEvent:Event):void {
  myXML = new XML(myEvent.target.data);
  myXML.ignoreWhite = true;

  autostart = parseInt(myXML.@autostart);
  rootDir = myXML.@rootDir;

  for(var i:uint=0; i<myXML.song.length(); i++) {
      theSongs[i] = myXML.song[i].@src;
  }
}
```

Listing 7.23: Die Funktion `EXMLComplete` *(Eventhandler für das Laden von Daten) enthält nun keine Fehlerbehandlung mehr.*

Die Fehlerbehandlung (mehrmaliges Laden von Daten, falls der Ladeversuch fehlgeschlagen hat) ist hierin nicht mehr definiert, da die gesamte Fehlerbehandlung in ActionScript 3.0 nun anders verläuft. Ein Fehler, der am wahrscheinlichsten auftritt, ist ein IOError – diesen gilt es abzufangen:

```
function EIOError(myEvent:IOErrorEvent):void {
  anzLadeversuche++;
  if(anzLadeversuche<maxLadeversuche) { myLoader.load(myRequest); }
  else {
      mcError.visible = true;
  }
}
myLoader.addEventListener(IOErrorEvent.IO_ERROR, EIOError);
```

Listing 7.24: Die Funktion `EIOError` *verarbeitet einen IOError. Bitte beachten Sie auch, dass die Eigenschaft* `visible` *nun ohne führendem Unterstrich „_" geschrieben wird.*

Schaltflächenereignisse: addEventListener

addEventListener **nun auch für Schaltflächen**

Da wir mit dem MovieClip `mcError` arbeiten und den Buttons btnJa und btnNein in diesem MovieClip `release`-Ereignisse zugewiesen haben, müssen wir uns noch um diese kümmern. In ActionScript 3.0 lassen sich Schaltflächenereignisse nicht mehr direkt zuweisen – vielmehr muss der Weg über `addEventListener` führen:

Aus:

```
var songTitel:String = "loading Playlist...";
var songInterpret:String = "";
var songAlbum:String = "";
var theTimeInfo:String = "--:--";

mcError.btnNein.onRelease = function():Void {
  _root.mcError._visible = false;
  _root.songTitel = "ERROR WHILE LOADING PLAYLIST";
}
mcError.btnJa.onRelease = function():Void {
  _root.myLoader.load(mySource);
  _root.mcError._visible = false;
}
```

wird:

```
Titel.text = "loading Playlist...";
Interpret.text = "";
Album.text = "";
Restzeit.text = "--:--";

function ENeinUp(myEvent:MouseEvent):void {
  mcError.visible = false;
  Titel.text = "ERROR WHILE LOADING PLAYLIST";
}
function EJaUp(myEvent:MouseEvent):void {
  myLoader.load(myRequest);
  mcError.visible = false;
}
mcError.btnNein.addEventListener(MouseEvent.MOUSE_UP, ENeinUp);
mcError.btnJa.addEventListener(MouseEvent.MOUSE_UP, EJaUp);
```

Listing 7.25: ActionScript 3.0 erfordert `addEventListener` *für die Zuweisung von Eventhandlern (auch bei Schaltflächen).*

Diese Vorgehensweise gilt im Übrigen selbstverständlich auch für sämtliche anderen Buttons, die wir im Audioplayer verwenden: Play, Stop, FWD, RWD.

Zugriff auf dynamische Textfelder über `text`

Die Variablen `songTitel`, `songAlbum` und `songInterpret` aus ActionScript 2.0 wurden durch den Zugriff auf die dynamischen Textfelder über deren Namen ersetzt, da der direkte Zugriff von dynamischen Textfeldern auf Variablen (zur Darstellung von deren Werten) nicht mehr möglich ist. Stattdessen greift man auf den Namen des Textfelds und dessen Eigenschaft `text` zu.

7.6.3 Änderungen im Soundobjekt

Grundsätzlich wurde die `Sound`-Klasse in das Paket `flash.media.Sound` verschoben. Jedoch ist dies bei weitem nicht die einzige Änderung, welche die neue ActionScript-Version mit sich bringt.

1. Laden von Sounddateien: Wie in ActionScript 3.0 üblich, erfolgt das gesamte Laden von Dateien über das `URLRequest`-Objekt. Somit wird beim Laden von Sounddateien einem Soundobjekt beim Laden eine Instanz eines `URLRequest`-Objekts zugewiesen.

2. Verwenden der `Sound`-Klasse: Diese wird zum Laden von Sounddateien, Verwalten allgemeiner Soundeigenschaften sowie zum Starten der Soundwiedergabe verwendet.

3. Verwenden der `SoundChannel`-Klasse: Sobald über die `Sound`-Klasse eine Sounddatei abgespielt wird, wird eine neue Instanz eines `SoundChannel`-Objekts erzeugt. Damit wird im Weiteren die Wiedergabe gesteuert.

Des Weiteren ist es nun möglich, auch gestreamte Sounds ab einer gewissen Position abspielen zu lassen – dies ermöglicht es uns, einen Pause-Button für Streaming-Sounds zu erstellen. (Dies ist in diesem Beispiel auch umgesetzt worden: Der Stop-Button wurde in einen Pause/Stop-Button umgewandelt – Details siehe weiter unten.)

7.6.4 Änderungen in der Abspielsteuerung

Die Abspielsteuerung betrifft die Buttons Play, Stop, RWD und FWD. Hier sind Änderungen an den Eventhandlern der Buttons vorzunehmen, da die direkte Zuweisung von Ereignissen in ActionScript 3.0 nicht mehr funktioniert. Somit erhalten wir anstatt:

```
btnPlay.onRelease = function():Void { playSong(0); }
btnStop.onRelease = function():Void { stopSong(); }
btnRWD.onRelease = function():Void { playSong(-1); }
btnFWD.onRelease = function():Void { playSong(1); }
```

Listing 7.26: ActionScript-2.0-Code ...

nun:

```
var isPaused:Boolean = false;
var isStopped:Boolean = false;
var pausingPos:Number = 0;

function EPlay(myEvent:MouseEvent):void { playSong(0); }
function EStop(myEvent:MouseEvent):void {
  if(!isStopped) {
      if(!isPaused) {
      isPaused = true;
      pausingPos = mySoundchannel.position;
      }
      else {
      isPaused = false;
      isStopped = true;
      pausingPos = 0;
      }
      stopSong();
  }
}
function ERWD(myEvent:MouseEvent):void { playSong(-1); }
function EFWD(myEvent:MouseEvent):void { playSong(1); }

btnPlay.addEventListener(MouseEvent.MOUSE_UP, EPlay);
btnStop.addEventListener(MouseEvent.MOUSE_UP, EStop);
btnFWD.addEventListener(MouseEvent.MOUSE_UP, EFWD);
btnRWD.addEventListener(MouseEvent.MOUSE_UP, ERWD);
```

Listing 7.27: ... im Gegensatz zu ActionScript-3.0-Code. Die Funktion `EStop` *musste erweitert werden, da der Stop-Button nun zwei Funktionen (Pause und Stopp) übernehmen soll.*

Wie Sie sehen, nennt sich das `onRelease`-Ereignis nun `Mouse_UP` und befindet sich in der Klasse `MouseEvent`. In der Funktion `EStop` finden Sie wesentlich mehr Code als im vergleichbaren Code in ActionScript 2.0. Dies hat hauptsächlich mit zwei Dingen zu tun:

1. Nachdem es in ActionScript 3.0 nun auch möglich ist, einen gestreamten Sound an einer gewissen Stelle starten zu lassen, kann ein „Pause"-Verhalten programmiert werden: Beim ersten Klick auf den Pause/Stop-Button wird der Song pausiert (bei Klick auf den Play-Button startet der Sound an der gestoppten Stelle), beim zweiten Klick wird er tatsächlich gestoppt und beginnt bei einem Klick auf den Play-Button von Neuem.

2. Da nun sowohl mit einem `Sound`- als auch mit einem `SoundChannel`-Objekt zu arbeiten ist, wird das Stoppen eines Sounds aufwändiger: Es muss einerseits (sofern der Datentransfer der Datei noch im Gange ist; betrifft die `Sound`-Instanz) der Datentransfer und andererseits das Abspielen des Songs (betrifft die `SoundChannel`-Instanz) gestoppt werden. Da nur ein laufender Transfer und nur ein gerade aktiver `SoundChannel` gestoppt werden kann, muss man sich – beispielsweise in Form einer Variable – „merken", ob eben gerade abgespielt wird. Eine Alternative zu dieser Vorgehensweise ist die Variante mit `try` & `catch`, die in der Datei *audioplayer_02a.fla* dargestellt ist.

Der Ablauf der `EStop`-Funktion ist also wie folgt:

- Sofern der Song nicht schon gestoppt ist (das wäre bei `isStopped==true`), wird überprüft ob der Song noch nicht pausiert ist (`isPaused==false`).
- Falls `isPaused==false` wird der Song „pausiert" (natürlich wird das Abspielen komplett gestoppt, aber die Position des Abstoppens wird in der Variable `pausingPos` gespeichert).
- Falls `isPaused==true` wird der Song gestoppt (dies bedeutet, dass `pausingPos=0` gesetzt wird).

Wie Sie sehen, spielt die Variable `pausingPos` eine wesentliche Rolle: Grundsätzlich wird jeder Song über diese Variable `pausingPos` (Anzahl der Millisekunden ab Beginn der Sounddatei) gestartet. Aus diesem Grund muss im Fall eines tatsächlich gestoppten Sounds die Variable `pausingPos` den Wert 0 erhalten.

Des Weiteren müssen die Funktionen `playSong` und `stopSong` entsprechend angepasst werden:

```
var mySound:Sound;
var mySoundchannel:SoundChannel;

function playSong(dir:Number):void {
  if(isStopped || dir!=0) { Restzeit.text = "--:--"; }
  Titel.text = "getting MP3-Information...";
  Album.text = "";
  Interpret.text = "";
```

```
    actualSong += dir;
    if(actualSong==theSongs.length) { actualSong = 0; }
    if(actualSong==-1) { actualSong = theSongs.length-1; }

    isPaused = false;
    isStopped = false;

    if(dir!=0) { pausingPos = 0; }

    if(mySound!=null && !isStopped) { stopSong(); }

    myRequest.url = rootDir+"/"+theSongs[actualSong];
    mySound = new Sound();
    mySound.addEventListener(Event.ID3, EID3);
    mySoundchannel = new SoundChannel();
    mySoundchannel.addEventListener(Event.SOUND_COMPLETE, ESoundComplete);

    mySound.load(myRequest);
    mySoundchannel = mySound.play(pausingPos);
  }

  function stopSong():void {
    mySoundchannel.stop();
    if(mySound.bytesLoaded!=mySound.bytesTotal) { mySound.close(); }
    clearInterval(itvTimer);
    if(isStopped) {
        Restzeit.text = "--:--";
        Titel.text = "(stopped)";
        Interpret.text = "";
        Album.text = "";
    }
  }
```

Listing 7.28: Die „neuen" Funktionen `playSong` *und* `stopSong`

Befassen wir uns zunächst mit der `playSong`-Funktion, welche die eingangs erzeugte `Sound`-Instanz `mySound` sowie die `SoundChannel`-Instanz `mySoundchannel` verwendet (diese wurde – um Verwechslungen mit der ActionScript-2.0-Variante zu vermeiden – von `mySong` auf `mySound` umgetauft):

- Zu Beginn werden die dynamischen Textfelder „zurückgesetzt".
- Danach wird der aktuell abzuspielende Song ermittelt (siehe hierzu auch die Erklärungen aus dem ActionScript-2.0-Teil: Listing 7.10).
- Da beim Aufruf dieser Funktion in jedem Fall ein Song abgespielt werden soll, werden die Variablen `isPaused` und `isStopped` auf den Wert `false` gesetzt: `isPaused = false; isStopped = false;`.
- Je nachdem, ob der aktuelle (`dir==0`) oder der nächste oder vorige Song abzuspielen ist, wird die Variable `pausingPos` auf 0 gesetzt, da nur im Fall des aktuellen Songs (Betätigen des Play-Buttons nach dem Betätigen des Pause/Stop-Buttons) ab einer gewissen Position abgespielt werden soll: `if(dir!=0) { pausingPos = 0; }`.
- In jedem Fall muss ein derzeit abgespielter Song beendet werden: `if(mySound!=null && !isStopped) { stopSong(); }`. Da dies nur geschehen kann, wenn bereits ein Soundobjekt existiert (also ungleich `null` ist) und ein Stoppen nur dann Sinn macht, wenn der Song noch nicht gestoppt ist, werden diese beiden Eigenschaften bzw. Variablen zuvor abgefragt.
- Nun folgt der eigentlich wichtige Teil beim Arbeiten mit Sounds:
  - Zunächst bedienen wir uns der bereits seit dem Laden der XML-Daten existenten `URLRequest`-Instanz `myRequest` und weisen der Eigenschaft `url` die URL zur Sounddatei aus dem Sound-Array zu: `myRequest.url = rootDir+"/"+theSongs[actualSong];`.
  - Danach wird eine `Sound`-Instanz gesetzt und dieser und der `SoundChannel`-Instanz ein Eventhandler (für das Ereignis `ID3`; Auslesen der ID3-Tags möglich) zugewiesen:

```
mySound = new Sound();
mySound.addEventListener(Event.ID3, EID3);
```

  - Zu guter Letzt wird das in der `URLRequest`-Instanz definierte MP3-File geladen, der `SoundChannel` mit dem Abspielen der Datei beauftragt und dem `SoundChannel` mitgeteilt, was nach dem erfolgten Abspielen der Sounddatei geschehen soll (Ereignis `SOUND_COMPLETE`; Abspielen der Sounddatei beendet):

```
mySound.load(myRequest);
mySoundchannel = mySound.play(pausingPos);
mySoundchannel.addEventListener(Event.SOUND_COMPLETE, ESoundComplete);
```

Fassen wir noch kurz zusammen:

1. Zum Laden von Daten (auch MP3-Files) wird generell das `URLRequest`-Objekt verwendet.
2. Das Soundobjekt erhält die Daten des `URLRequest`-Objekts und „verwaltet" diese sozusagen. Im Gegensatz zu ActionScript 2.0 wird das Soundobjekt nun nicht mehr zum Abspielen des Sound-File verwendet.
3. Zum Steuern der Sounddaten wird das `SoundChannel`-Objekt verwendet.

Bleibt noch die Funktion `stopSound`:

- Im ersten Schritt wird das Abspielen in der SoundChannel-Instanz `mySoundchannel` gestoppt: `mySoundchannel.stop();`.
- Sollte in der Soundinstanz `mySound` noch ein Datentransfer aktiv sein (das wäre der Fall, wenn die geladenen Bytes der MP3-Datei noch nicht den zu übertragenen Bytes entsprechen), wird dieser ebenfalls beendet: `if(mySound.bytesLoaded!=mySound.bytesTotal) { mySound.close(); }`.
- Danach wird das gerade laufende Intervall zum Anzeigen der Restzeit gestoppt: `clearInterval(itvTimer);`.
- Falls der Sound nicht pausiert, sondern gestoppt wird, werden zusätzlich noch die Textfelder zurückgesetzt.

Alternative: try & catch in der Funktion stopSound

```
function stopSong():void {
  try { mySoundchannel.stop(); }
  catch(myError:Error) { trace("SoundChannel kann nicht gestoppt werden.
  Error="+myError.message); }
  try { mySound.close(); }
  catch(myError:Error) { trace("Soundtransfer kann nicht geschlossen
  werden, da er wahrscheinlich bereits abgeschlossen ist.
  Error="+myError.message); }
  clearInterval(itvTimer);
  if(isStopped) {
      Restzeit.text = "--:--";
      Titel.text = "(stopped)";
      Interpret.text = "";
      Album.text = "";
  }
}
```

Listing 7.29: Alternative Abarbeitung der `stopSound`*-Funktion in der Datei* audioplayer_02a.fla

Anstatt die SoundChannel-Instanz mySoundchannel „einfach so“ zu stoppen, wird dies mittels try versucht. Sollte der Versuch nicht klappen (was nicht tragisch wäre, sondern nur die Info, dass das Abspielen bereits gestoppt wurde), kann man im catch-Teil den Fehler ausgeben – unbedingt notwendig ist das jedoch nicht.

Dasselbe Prozedere durchläuft der Versuch, die Sound-Instanz mySound in ihrer Übertragung zu beenden.

Kommen wir zum Auslesen der ID3-Tags sowie dem Timer für die Restzeitanzeige. Das Auslesen der ID3-Tags erfolgt nach wie vor im Sound-Objekt (wie in der Beschreibung der Funktion playSong beschrieben):

```
var itvTimer:Number;
function EID3(myEvent:Event):void {
  Interpret.text = mySound.id3.TPE1;
  Album.text = mySound.id3.TALB;
  Titel.text = mySound.id3.TIT2;
  itvTimer = setInterval(showTimer,1000);
}

function showTimer():void {
  var theLength:Number = Math.floor((mySound.id3.TLEN-mySoundchannel.
  position)/1000);
  var min:Number = Math.floor(theLength/60);
  var sek:String = "";
  if(theLength-60*min<10) { sek += "0"; }
  sek += theLength-60*min;
  Restzeit.text = "-"+min+":"+sek;
}
```

Listing 7.30: Die Funktion EID3 *wird zum Anzeigen der ID3-Tags einer MP3-Datei benötigt, die Funktion* showTimer *dient zur Anzeige der Restzeit eines abspielenden Songs.*

Diese beiden Funktion sind relativ straight forward:

- Die Funktion EID3 dient als Eventhandler und wird aufgerufen, sobald eine Sound-Instanz die ID3-Tags aus einer MP3-Datei auslesen konnte, wobei wir uns hier lediglich auf die ID3-Tags der Version 2 beziehen. Sie liest die Tags TPE1 (Interpret), TALB (Album) und TIT2 (Titel) der MP3-Datei aus. Sobald dies geschehen ist, wird ein Intervall gesetzt, das in regelmäßigen Abständen (alle 1000 Millisekunden, also jede Sekunde) die Funktion showTimer aufruft.

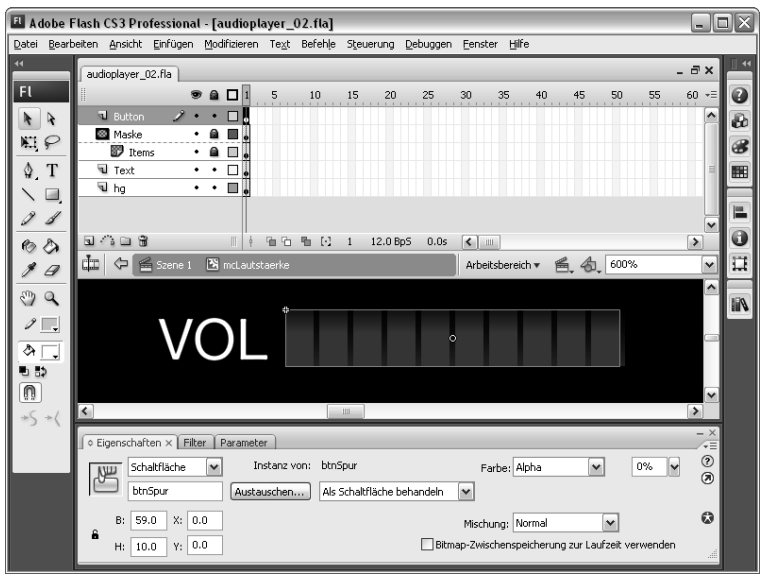
- Die Funktion `showTimer` errechnet aus der Länge der MP3-Datei (`TLEN`; in Millisekunden) und der aktuellen Abspielposition (`mySoundchannel.position`; in Millisekunden) die aktuelle Restzeit in Sekunden. Daraus werden die Minuten und die Sekunden für die Anzeige berechnet, wobei den Sekunden eine führende Null angehängt wird, sollten die Sekunden weniger als zehn sein. Zum Schluss wird aus den berechneten Werten ein String der Form „-MM:SS“ gebastelt und dem Textfeld `Restzeit` zugewiesen.

7.6.5 Änderungen in der Lautstärkeregelung

Die letzte Änderung betrifft die Lautstärkeregelung. Da MovieClips in ActionScript 3.0 keine Button-Ereignisse mehr zugewiesen werden können, muss dem Lautstärkeregler im Design ein unsichtbarer Button hinzugefügt werden, den der User dann anklicken kann. Dieser Button wird die Größe der Spur des Hintergrunds haben – somit bekommt der User den Eindruck, er könnte direkt den Lautstärkeregler anklicken:

Abbildung 7.6

Der Lautstärkeregler erhält einen zusätzlichen unsichtbaren Button namens `btnSpur`.



Die Änderungen bezüglich des Lautstärkereglers ergeben sich dann wie nachfolgend. Aus:

```
var itvLautstaerke:Number;
mcLautstaerke.mcSpur.onPress = function():Void {
  itvLautstaerke = setInterval(setLautstaerke,50);
}
mcLautstaerke.mcSpur.onRelease = function():Void {
  clearInterval(itvLautstaerke);
```

```
}
mcLautstaerke.mcSpur.onReleaseOutside = function():Void {
  clearInterval(itvLautstaerke);
}
```

wird:

```
var itvLautstaerke:Number;
function ELautstaerkePress(myEvent:MouseEvent):void {
  itvLautstaerke = setInterval(setLautstaerke,50);
}
function ELautstaerkeRelease(myEvent:MouseEvent):void {
  clearInterval(itvLautstaerke);
}
function ELautstaerkeRollOutside(myEvent:MouseEvent):void {
  clearInterval(itvLautstaerke);
}
mcLautstaerke.btnSpur.addEventListener(MouseEvent.MOUSE_DOWN,
ELautstaerkePress);
mcLautstaerke.btnSpur.addEventListener(MouseEvent.MOUSE_UP,
ELautstaerkeRelease);
mcLautstaerke.btnSpur.addEventListener(MouseEvent.ROLL_OUT,
ELautstaerkeRollOutside);
```

Listing 7.31: Die neue Ereignisbehandlung für den Lautstärkeregler

Letzten Endes hat sich lediglich der Aufruf der Eventhandler geändert, wobei uns das schon aus früheren Überlegungen bekannt ist. Da das Ereignis `onReleaseOutside` nicht mehr existiert, muss man auf das Ereignis `ROLL_OUT` zurückgreifen: Dieses Ereignis tritt auf, sobald der User mit der Maus aus einem bestimmten MovieClip herausrollt.

Die Funktion `setLautstaerke` hat sich gegenüber der ActionScript-2.0-Variante ebenfalls deutlich geändert. Aus:

```
function setLautstaerke():Void {
  var mousex:Number = mcLautstaerke._xmouse;
  var maxX:Number = mcLautstaerke.mcSpur._width;
  mcLautstaerke.mcMaske._width = mousex;
  mySong.setVolume(mousex/maxX*100);
}
```

wird:

```
function setLautstaerke():void {
  if(mySoundchannel!=null) {
      var mousex:Number = mcLautstaerke.mouseX;
      var maxX:Number = mcLautstaerke.mcSpur.width;
      mcLautstaerke.mcMaske.width = mousex;

      var myTransform:SoundTransform = mySoundchannel.soundTransform;
      myTransform.volume = mousex/maxX;
      mySoundchannel.soundTransform = myTransform;
  }
}
```

Listing 7.32: Um die Lautstärke einer `SoundChannel`*-Instanz zu verändern, bedarf es einer* `SoundTransform`*-Instanz.*

Grundsätzlich kann eine Veränderung der Lautstärke nur dann erfolgen, wenn es eine `SoundChannel`-Instanz gibt – in unserem Fall trägt diese den Namen `mySoundchannel`. Ist diese vorhanden, werden x- und y-Position der Maus innerhalb des MovieClips `mcLautstaerke` ausgelesen und die Maske zum Anzeigen der Lautstärkestriche wird entsprechend gesetzt.

Danach erfolgt der erste wichtige Schritt: die derzeitigen `SoundTransform`-Einstellungen werden aus der `SoundChannel`-Instanz `mySoundchannel` ausgelesen und der `SoundTransform`-Instanz `myTransform` zugewiesen:

```
var myTransform:SoundTransform = mySoundchannel.soundTransform;
```

Danach wird in dieser gerade erzeugten Instanz der Wert der Eigenschaft `volume` auf den entsprechenden Wert gesetzt. Bitte beachten Sie an dieser Stelle, dass die Eigenschaft `volume` nun nicht mehr in Prozent, sondern in Werten zwischen 0 und 1 angegeben werden muss:

```
myTransform.volume = mousex/maxX;
```

Ist dies erfolgt, schreibt man die geänderten Einstellungen wieder zurück in die `SoundChannel`-Instanz:

```
mySoundchannel.soundTransform = myTransform;
```

7.6.6 Songs aus einer ComboBox abspielen

Als Erweiterung des aktuellen Players könnte man beispielsweise eine ComboBox mit allen Songs der Playlist füllen und durch Auswählen eines Songs aus der ComboBox

diesen direkt abspielen. Gesagt – getan: Ziehen Sie hierzu eine ComboBox aus den Komponenten auf die Bühne und geben Sie dieser den Namen „myPlaylist“.

Nachdem dies erledigt ist, muss die Funktion `EXMLComplete` vor dem Aufruf der Funktion `initPlayer` wie folgt erweitert werden, um die ComboBox mit Inhalt zu füllen:

```
    ...
    myPlaylist.addEventListener(Event.CHANGE, EChange);
    myPlaylist.addItem({ label:"Bitte wählen Sie:", data:-1 });
    for(i=0; i<myXML.song.length(); i++) {
        myPlaylist.addItem({ label:myXML.song[i].@src, data:i });
    }

    initPlayer();
}
```

Listing 7.33: Befüllen der ComboBox `myPlaylist` *mit den MP3-Dateien aus der Playlist*

Bevor die `for`-Schleife das Befüllen der ComboBox übernimmt, wird noch ein Eintrag mit der Beschriftung „Bitte wählen Sie“ in die ComboBox geschrieben. Der zugehörige Wert dieses Eintrags ist -1 (dieser Wert wird später benötigt, da die Wahl des Users auch auf diesen Eintrag fallen kann, jedoch dann kein Song geladen werden soll).

Innerhalb der `for`-Schleife werden alle Einträge der XML-Datei-Tags `<song>` als Beschriftung (`label`) in die ComboBox geschrieben. Die zugehörigen Werte (`data`) der Einträge sind die fortlaufenden Zahlen der `for`-Schleife (und somit der Index innerhalb der Playlist `theSongs`).

Der Eventhandler `EChange` bestimmt, was beim Auswählen eines Songs aus der ComboBox geschehen soll:

```
function EChange(myEvent:Event):void {
  if(myPlaylist.selectedItem.data!=-1) {
      var dir:Number = myPlaylist.selectedItem.data-actualSong;
      playSong(dir);
  }
}
```

Listing 7.34: Der Eventhandler `EChange`*, der aufgerufen wird, sobald der User eine neue Auswahl in der ComboBox getroffen hat*

Sollte die Auswahl einen Wert ungleich -1 haben (-1 war der Wert für den Eintrag „Bitte wählen Sie:“), so wird in der Variable `dir` die Anzahl der Songs ermittelt, um

die in der Playlist zurück (falls kleiner 0) oder vor (falls größer 0) gesprungen werden muss. Dieser Wert wird dann an die Funktion `playSong` übergeben – fertig.

7.7 Mögliche Erweiterungen

Neben der Abspielsteuerung in Flash kann selbstverständlich auch eine Abspielsteuerung in XHTML erfolgen. Dabei müssen in XHTML entsprechende Buttons angelegt werden, die in Flash freigegebene Funktionen aufrufen (wie Sie das realisieren können, erfahren Sie im Kapitel *Clientseitiger Datenaustausch*).

Eine andere Erweiterung wäre etwa, dass per AJAX in regelmäßigen Abständen in der Datenbank nachgesehen wird, ob sich an der Playlist etwas verändert hat – wäre dies der Fall, so könnte die Playlist in Flash nachgeladen werden. Ein entsprechendes Beispiel finden Sie im Anschluss: der Videoplayer. Viel Spaß dabei!