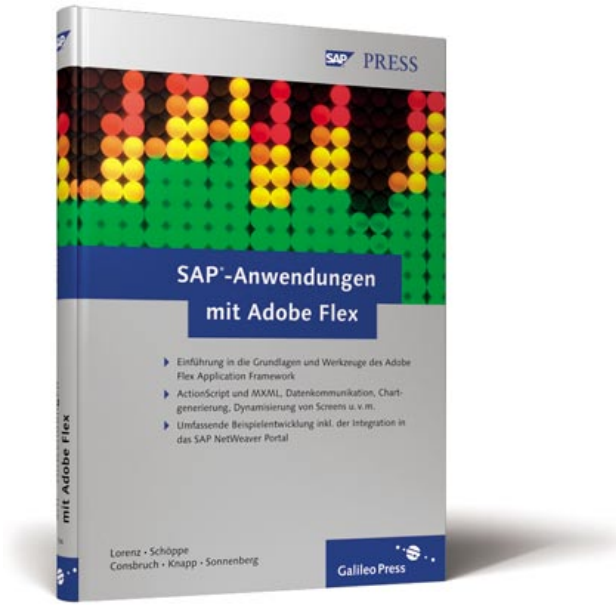
Armin Lorenz, Dr. Gunther Schöppe, Felix Consbruch,
Daniel Knapp, Frank Sonnenberg

SAP®-Anwendungen mit Adobe Flex



SAP PRESS

Auf einen Blick

Inhalt

Hinweis

In Anhang A werden die kompletten Listings der in diesem Kapitel dargestellten Dateien dargestellt. Sie finden sie auch auf der Webseite zum Buch unter *http://www.sap-press.de/1246* zum Download.

7.3.1 Projektstruktur des Frontends der Applikation

Das Frontend des Dispositions- und Zeiterfassungs-Tools wird im Adobe Flex Builder 2 entwickelt. Nach dem Öffnen des Flex Builder steht dem Benutzer eine auf Eclipse basierte und sehr leistungsstarke Entwicklungsumgebung zur Verfügung.

Projektdateien

Im ersten Schritt werden die notwendigen Dateien für unser Projekt definiert. Dieses beinhaltet das Flex-Projekt, die darin enthaltene MXML-Applikation, die dazugehörigen MXML-Components sowie die ActionScript-Dateien, die die Logik der Applikation enthalten werden.

Zum Start des Adobe Flex Builder wählen Sie diesen über das Windows-Startmenü aus – normalerweise wird er in **Start · Programme · Adobe · Adobe Flex Builder** installiert. Nach erfolgreichem Start legen Sie ein neues Projekt über **File · New · Flex Project** an (siehe Abbildung 7.50).

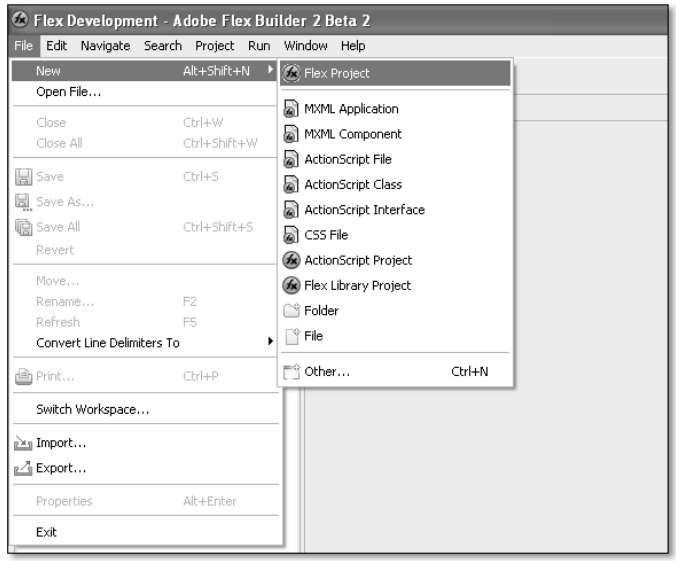


Abbildung 7.50 Erstellen des Flex-Projekts

Wahl des Flex-Servers

Es öffnet sich ein weiterer Dialog, in dem Sie Einstellungen zum Projekt vornehmen können (siehe Abbildung 7.51). Im ersten Schritt müssen Sie dazu den Flex-Server wählen, auf dem die Applikation erstellt wird.

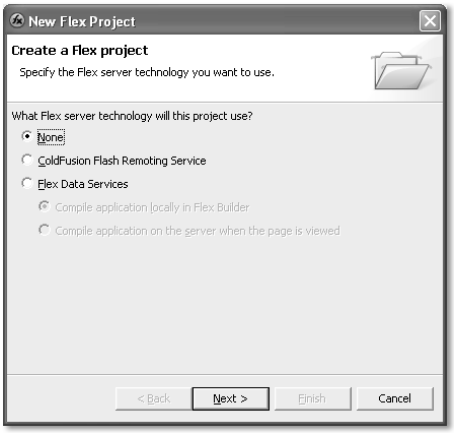


Abbildung 7.51 Auswahl des Flex-Servers

- Die Option **ColdFusion Flash Remoting Service** ist eine weiterführende Applikation, die die Flex Data Services 2 und ColdFusion MX 7.1 als Voraussetzung benötigt. Diese Art Applikation wird benötigt, um Flex- und ColdFusion-Applikationen miteinander zu verbinden.
- **Flex Data Services** sind um eine neue Architektur (Data Service Architecture) und Charting-Komponenten erweiterte Applikationen. Dadurch ist es möglich, weitere Technologien an Flex anzubinden (siehe Abbildung 7.52).

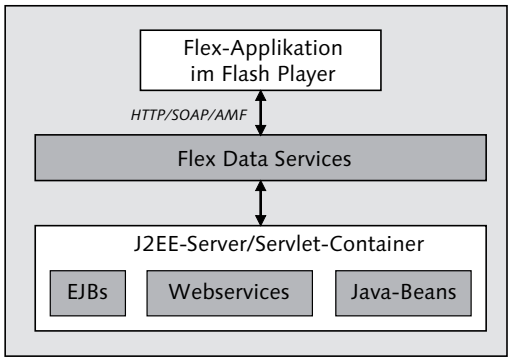


Abbildung 7.52 Flex Data Services

Da in unserem Fall kein Flex-Server installiert ist, wählen Sie hier die Einstellung **None** und fahren fort. Dies ist die Standardeinstellung für das Erstellen von Flex-Projekten.

Eindeutige Nomenklatur

Nun müssen Sie einen Namen an das Projekt vergeben. Wählen Sie hier den gleichen Namen wie auch für das Paket in der ABAP Workbench (Z_DZT_BOOK), um eine eindeutige Nomenklatur zu erreichen (siehe Abbildung 7.53).

Die Einstellung **Use default location** kann beibehalten werden: Sie gibt an, in welchem Verzeichnis sich die späteren Dateien befinden werden. Wahlweise kann auch ein anderes Verzeichnis für das Projekt genutzt werden.

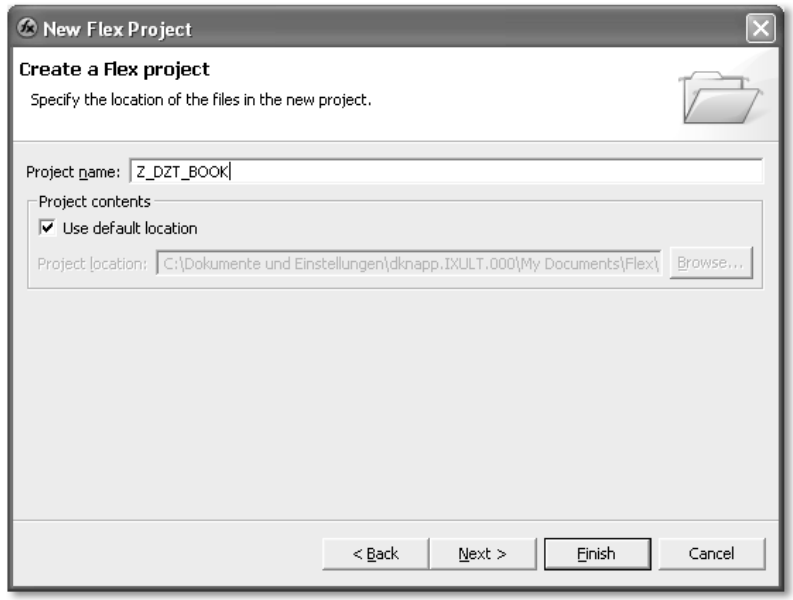


Abbildung 7.53 Vergabe des Projektnamens

Nachdem der Name des Projekts feststeht, können Sie weitere Pakete und Verzeichnisse anlegen, die im späteren Verlauf benutzt werden sollen. Im ersten Schritt legen Sie hierzu einen **Source folder** namens `src` an, der die MXML-Dateien enthalten soll (siehe Abbildung 7.54).

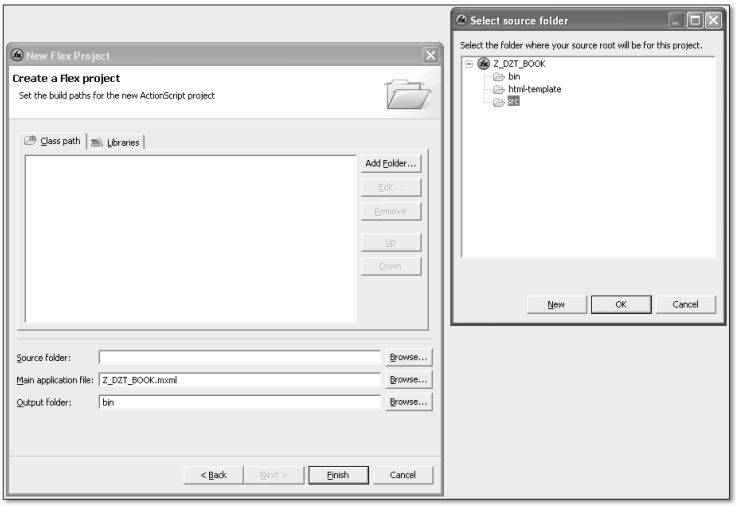


Abbildung 7.54 Anlegen von Verzeichnissen

Anschließend können Sie das Anlegen des Flex-Projekts mit dem **Finish**-Button abschließen.

Hinweis

Manchmal kann es passieren, dass eine Fehlermeldung im Statusfenster erscheint, die auf eine fehlende Datei verweist (`...frameworks/locale/de_de/framework_rb.swc`). Zur Lösung des Problems können Sie das Verzeichnis `frameworks/locale/en_US` in `de_de` kopieren (siehe Abbildung 7.55).

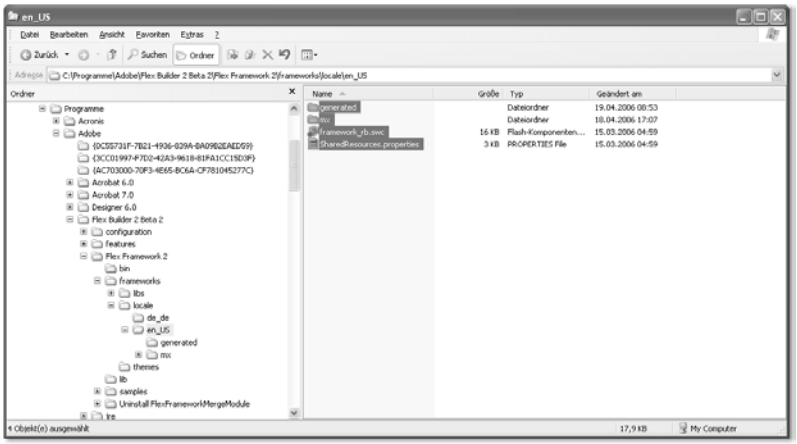


Abbildung 7.55 Verzeichniss en_US

ActionScript-Verzeichnis

Nachdem das Projekt angelegt wurde, können Sie mit der Entwicklung beginnen. Wir legen allerdings zuerst weitere Dateien an, um das Grundgerüst der Applikation zu definieren.

Klicken Sie mit der rechten Maustaste auf den Projektnamen, um über **New · Folder** einen neuen Ordner zu erstellen (siehe Abbildung 7.56). In unserem Fall legen wir den Ordner `as` an, der die ActionScript-Dateien enthalten soll.

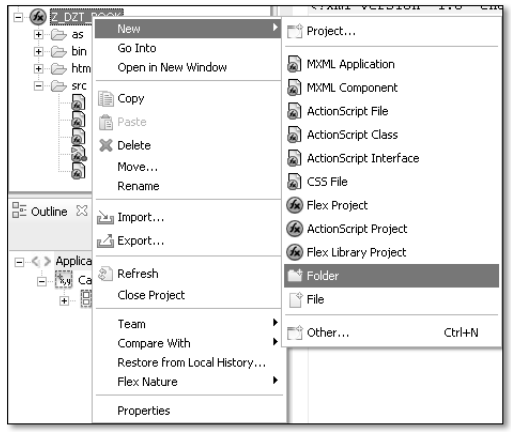


Abbildung 7.56 Anlegen eines neuen Verzeichnisses

Anlegen der MXML-Dateien

Im `src`-Verzeichnis des Projekts, das bei der Definition festgelegt wurde, befindet sich bereits die Datei `Z_DZT_BOOK.mxml` – die Hauptseite der Applikation. In der Hauptseite – und nur dort – befindet sich das `<mx:Application>`-Tag mit der zugehörigen Namespace-Deklaration.

Da unsere Applikation aus mehreren Teilbereichen bestehen wird, ist es sinnvoll, diese in einzelne MXML-Components aufzugliedern. Es werden nun nacheinander die folgenden Dateien angelegt:

- `dispo.mxml`
- `report.mxml`
- `tabcontrol.mxml`
- `ze.mxml`

Dies erfolgt, indem Sie mit der rechten Maustaste auf den **src**-Button klicken und **MXML Component** auswählen. Anschließend sollte das Projektverzeichnis Abbildung 7.57 gleichen.

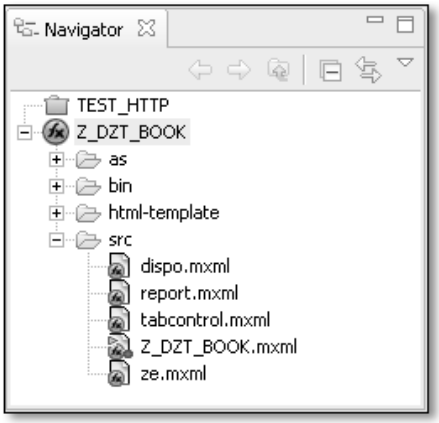


Abbildung 7.57 Angelegte MXML-Dateien

ActionScript-Dateien

Das Verzeichnis `as`, in dem die ActionScript-Dateien abgelegt werden sollen, wird auf analoge Weise definiert. Für unsere Applikation benötigen Sie folgende Dateien:

- `application.as`
- `dispo.as`
- `global.as`
- `report.as`
- `ze.as`

Das fertige Projektverzeichnis hat dann das in Abbildung 7.58 gezeigte Format.

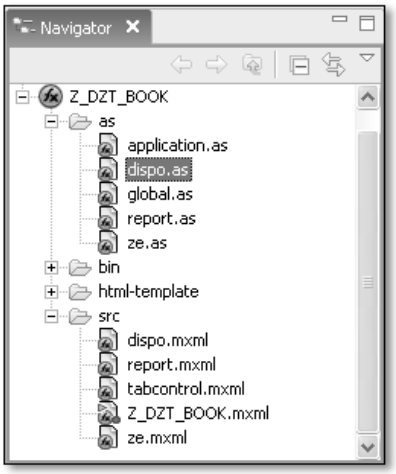


Abbildung 7.58 Projektverzeichnis nach dem Anlegen der Dateien

7.3.2 Grundgerüst des Frontends

Tabstrips

Die Beispielapplikation soll über Tabstrips realisiert werden, d.h. die Unterpunkte **Disposition**, **Zeiterfassung** und **Reporting** werden in eigenen Registern dargestellt. Tabstrips sind eine sehr praktikable Technik, um viele Informationen auf einer Seite übersichtlich darstellen zu können.

Die Hauptseite gliedert sich in den Log-in-Bereich sowie die Tabstrips auf, die in der Datei `tabcontrol.mxml` gehalten werden. Die logische Gliederung der Datei ist in Abbildung 7.59 dargestellt.

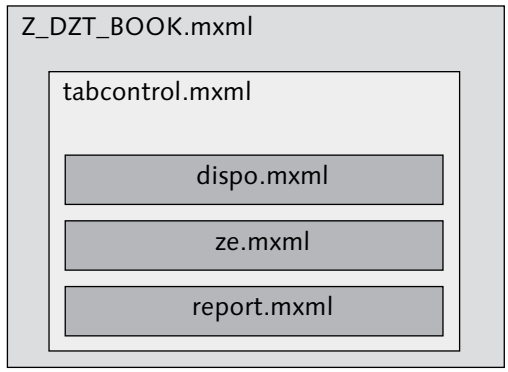


Abbildung 7.59 Logische Struktur der Dateien

Diese grobe Struktur können Sie wie in Listing 7.31 in MXML umsetzen: Um diesen Quelltext in die Applikation einzufügen, wechseln Sie von der **Design**-Sicht in die **Source**-Sicht.

```
<mx:Application
   xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
   layout="absolute" xmlns:comp="*"
   width="900" height="700">

   <mx:Canvas id="cv_login">
      <mx:VBox>
         <!-- Login-Bereich -->
      </mx:VBox>

   </mx:Canvas>
   <mx:Canvas id="cv_tabs">

      <mx:VBox>
         <comp:tabcontrol/>
```

```
      </mx:VBox>

   </mx:Canvas>
</mx:Application>
```

Listing 7.31 Applikationsdatei Z_DZT_BOOK.mxml

Anlage des Namensraumes

Der Namensraum `comp` wird angelegt, um auf Komponenten zuzugreifen. Zwar ist es nicht notwendig, hierfür einen extra Namensraum zu reservieren, allerdings erhöht dies die Übersichtlichkeit des Quelltexts beträchtlich. Die Applikation ist untergliedert in zwei `<mx:Canvas>`-Bereiche, die den Log-in-Bereich respektive die `Tabstrip`-Komponente beinhalten. Die Größe der Applikation wird zentral im `<mx:Application>`-Tag definiert und ist hier mit 900 mal 700 Pixeln eingetragen.

Die Komponente `tabcontrol.mxml` enthält weiterhin die einzelnen Tabstrips **Disposition**, **Zeiterfassung** und **Reporting**, die in Listing 7.32 auszugsweise dargestellt sind.

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

   <mx:VBox>
      <mx:Label text="Dispositionsmanager"
          fontFamily="Arial" fontSize="20"
          fontWeight="bold"/>

<mx:TabNavigator height="650" width="850">
         <comp:report label="Reporting"/>
         <comp:dispo  label="Disposition"/>
         <comp:ze     label="Zeiterfassung"/>
      </mx:TabNavigator>
   </mx:VBox>

</mx:Canvas>
```

Listing 7.32 Auszug aus der MXML-Komponente tabcontrol.mxml

Überschriften der Registerkarten

Wie man sieht, wird hier kein `<mx:Application>`-Tag eingebunden, da es sich um eine Komponente und nicht die Hauptdatei handelt. Über das `label`-Tag der einzelnen Komponenten im `<mx:TabNavigator>`-Tag werden die Überschriften der Registerkarten definiert, das `<mx:Label>`-Tag darüber gibt die Überschrift an.

Die so definierte Applikation kann bereits im Design-Modus des Flex Builder angezeigt werden, wobei die Komponenten noch leer sind. In Abbildung 7.60 sind die Überschriften der Registerkarten sowie die in der Datei `tabcontrol.mxml` definierte Überschrift (**Dispositionsmanager**) zu sehen.

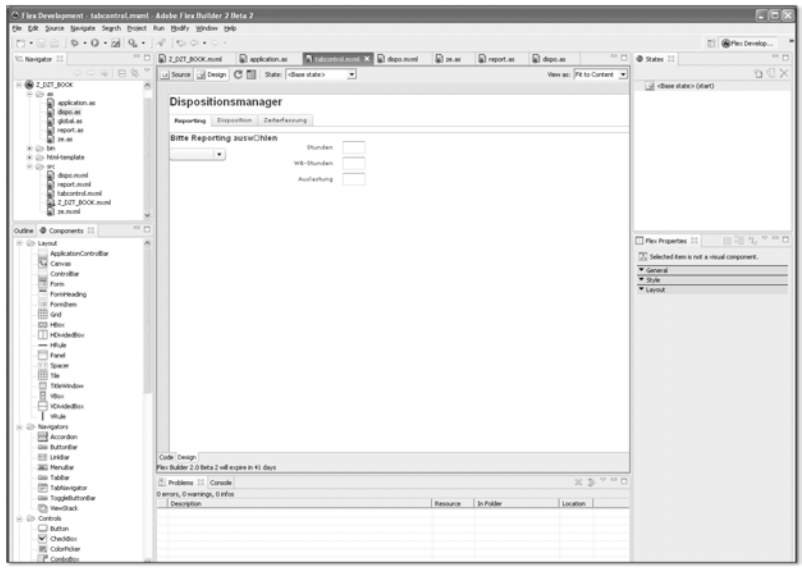


Abbildung 7.60 Datei tabcontrol.mxml im Design-Modus im Flex Builder

Die komplette Datei `tabcontrol.mxml` ist in Listing 7.33 dargestellt. Sie enthält noch einen kurzen Skriptteil, der Daten aus der Hauptapplikation entgegennehmen kann und diese an die Komponenten verteilt. Es wird im Verlauf dieses Kapitels weiter darauf eingegangen.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns="*" xmlns:comp="*">

   <mx:Script>
      <![CDATA[
         [Bindable]
         public var tab_pernr:String;

         [Bindable]
         public var tab_url_dispo:String;
```

```
        [Bindable]
        public var tab_url_ze:String;
      ]]>
   </mx:Script>
   <mx:VBox>
      <mx:Label text="Dispositionsmanager"
          fontFamily="Arial" fontSize="20"
          fontWeight="bold"/>

      <!-- Registerkartendarstellung und Einbindung der
           Komponenten -->
      <mx:TabNavigator height="650" width="850">
         <comp:report label="Reporting"
               l_pernr="{tab_pernr}"/>
         <comp:dispo  label="Disposition"
               l_pernr="{tab_pernr}"
               l_url="{tab_url_dispo}"/>
         <comp:ze label="Zeiterfassung"
               l_pernr="{tab_pernr}"
               l_url="{tab_url_ze}"/>
      </mx:TabNavigator>
   </mx:VBox>
</mx:Canvas>
```

Listing 7.33 Datei tabcontrol.mxml

Das Grundgerüst der Applikation ist damit definiert, und Sie können es zur weiteren Entwicklung verwenden.

7.3.3 Entwicklung der Dispositionskomponente

Die Disposition – wie auch die Zeiterfassung – werden als *Erfassungskomponenten* bezeichnet. In diesem Abschnitt wird Schritt für Schritt erläutert, wie die Disposition dargestellt und an das Backend angeschlossen werden kann, Abschnitt 7.3.4 zeigt das Vorgehen zur Zeiterfassungskomponente.

Struktur der Disposition

Wir wollen die Disposition so gestalten, dass auf der linken Seite Dispositionen erfasst werden können, die auf der rechten Seite dargestellt werden. Ziel ist die Entwicklung der Dispositionskomponente mit dem Layout aus Abbildung 7.61.

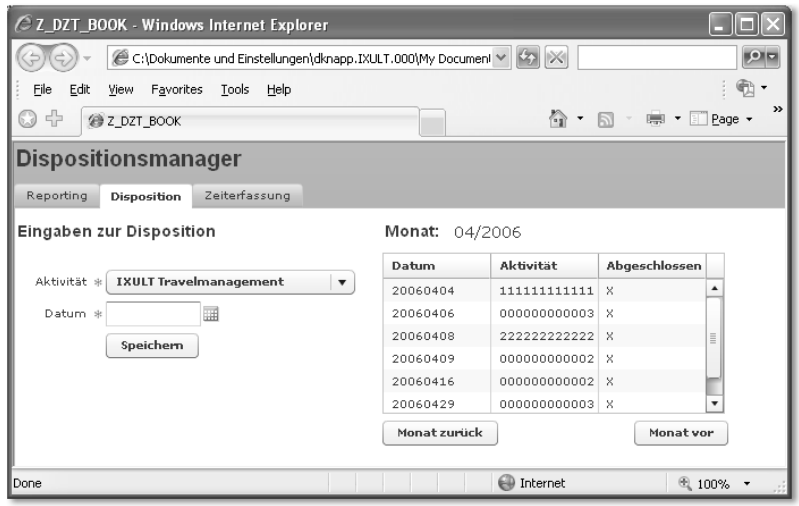


Abbildung 7.61 Darstellung der Dispositionsseite

Layout-Komponenten

Grundsätzlich ordnet Flex einzelne Komponenten immer untereinander an, wenn sie in das Layout gezogen werden. Um das zu verhindern, müssen Komponenten in Layoutbereiche gekapselt werden: `<mx:HBox>` und `<mx:VBox>`. Die `HBox` gruppiert Elemente in der horizontalen Ebene, die `VBox` in der vertikalen.

Wie Sie in Abbildung 7.61 sehen, ist die Disposition in zwei Bereiche unterteilt, der linke zur Eingabe und der rechte zur Ausgabe. Diese beiden Bereiche werden über ein `<mx:HBox>`-Tag voneinander getrennt:

```
<mx:HBox>
   <!-- Linke Seite -->
   <!-- Rechte Seite -->
</mx:HBox>
```

Die zur Eingabe notwendigen Felder sind zum einen die Überschrift **Eingaben zur Disposition** sowie die Eingabefelder und der **Speichern**-Button, die allesamt übereinander dargestellt werden müssen. Dazu ist eine Aufgliederung der Elemente der linken Seite in ein `<mx:VBox>`-Tag notwendig. Das Gleiche gilt für die rechte Seite, die aus dem Monat, einer Tabelle und zwei Buttons besteht. Das folgende Listing 7.34 zeigt die Struktur mit diesen Layoutelementen.

```
<mx:HBox>
   <!-- Linke Seite: Eingabe der Disposition -->
   <mx:VBox>
      <!-- Eingabeelemente -->
   </mx:VBox>

   <!-- Rechte Seite: Anzeige der Dispositionen -->
   <mx:VBox>
      <!-- Darstellungselemente -->
   </mx:VBox>
</mx:HBox>
```

Listing 7.34 Layoutelemente der Datei dispo.mxml

Linke Seite der Disposition

Nun können Sie die einzelnen Seiten mit funktionalen Elementen bestücken: Nach der Definition des Funktionsbausteins zum Hinzufügen von Dispositionen (siehe Abschnitt 7.2.3) werden die Personalnummer, das Datum und die Aktivität benötigt. Die Personalnummer wird dabei beim Einloggen festgelegt, sodass der Anwender nur noch ein Datum und die entsprechende Aktivität auswählen muss.

<mx:Form>

Flex bietet für diesen Fall eine einfache Weise an, die so genannten *Form-Elemente* einzubinden – über das `<mx:Form>`-Tag. Das `<mx:Form>`-Tag wird in einzelne `<mx:FormItem>`-Tags untergliedert, die wiederum die eigentlichen Elemente enthalten. Dadurch ist eine strukturierte Aufteilung der Komponenten möglich.

Für die Disposition werden drei Komponenten benötigt:

- Die Komponente `<mx:ComboBox>` dient zur Auflistung der Aktivitäten als Drop-down-Liste.
- Die Komponente `<mx:DateField>` bietet eine komfortable Art, ein Datum einzubinden.
- Die Komponente `<mx:Button>` wird den **Submit**-Button der Disposition darstellen.

Das `<mx:Form>`-Tag der Disposition ist in Listing 7.35 gezeigt.

```
<mx:Form id="form_dispo">
   <mx:FormItem label="Aktivität" required="true">
      <mx:ComboBox id="form_activity"
          dataProvider="" labelField="value"/>
```

```
    </mx:FormItem>
    <mx:FormItem label="Datum" required="true">
       <mx:DateField id="form_date"
           labelFunction="df_convert"/>
    </mx:FormItem>
    <mx:FormItem>
       <mx:Button id="form_submit" label="Speichern"
          click="add_disposition()"/>
    </mx:FormItem>
</mx:Form>
```

Listing 7.35 Form-Elemente der Disposition

FormItem-Aktivität

Das erste `FormItem`-Element dient zur Darstellung der Aktivität und enthält das entsprechende Label. Die Eigenschaft `required` gibt an, ob das Item ein Mussfeld ist oder nicht.

Hinweis

Die Eigenschaft `required` ist nur eine Darstellungsvariante und zeigt einen roten Stern (*) neben dem Feld. Es wird nicht überprüft, ob das Feld tatsächlich gefüllt wurde.

Das `FormItem`-Element selbst enthält die `ComboBox`, die die Aktivitäten darstellen soll. Die Aktivitäten stammen dabei aus dem Frontend und werden über die Datei `data.xml` an das Frontend übermittelt. Das Füllen der `ComboBox` übernimmt die Eigenschaft `dataProvider`, auf die im Laufe des Kapitels noch eingegangen wird.

Wichtig ist, dass Sie die Eigenschaft `labelField` setzen, um zu kennzeichnen, welches Element der XML-Struktur später dargestellt werden soll.

FormItem-Datum

Das zweite `FormItem`-Element stellt das Datum dar und wird ebenfalls mit der Eigenschaft `required` bestückt. Das `FormItem` enthält als Kind das Tag `<mx:DateField>`, dem neben der ID eine `labelFunction` zugewiesen wird. `labelFunction` dient dazu, ein Datum erst nach eigenen Wünschen zu formatieren, bevor es dargestellt wird. Die Funktion `df_convert()` wird später in der Datei `global.as` implementiert.

Das `DateField` ist eine Komponente, die aus einem Eingabefeld und einem Button in Form eines kleinen Kalenders besteht. Klickt man auf diesen Button, wird ein Kalender geöffnet, der die Eingabe des Datums erleichtert (siehe Abbildung 7.62).

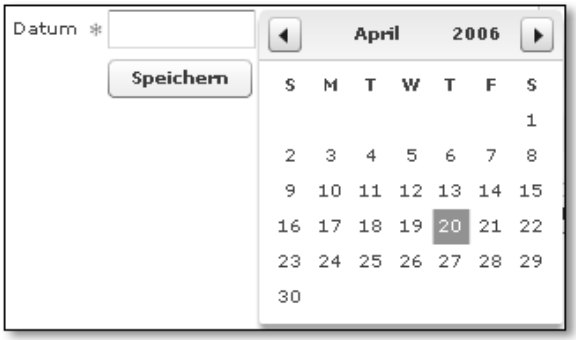


Abbildung 7.62 Eingabehilfe eines Datums im DateField

FormItem-Button

Um die getätigten Eingaben an das Backend zu übermitteln, benötigen Sie einen Button. Dieser Button wird über das dritte `FormItem`-Element eingebunden und enthält das Tag `<mx:Button>`. Diesem Tag können Sie ein Event zuweisen – in diesem Fall das System-Event `click` –, das eine Funktion zur Bearbeitung erwartet. Die Funktion `add_disposition()`, die das Event entgegennimmt, wird Bestandteil der Datei `dispo.as` werden.

Das so festgelegte Dispositions-Coding hat das Layout aus Abbildung 7.63 zur Folge.

Abbildung 7.63 Layout des Form-Tags

Rechte Seite der Disposition

Einbinden von Tabellen

Die rechte Seite der Disposition dient zur Darstellung der getätigten Eingaben in einer Tabelle für einen Monat. Tabellen werden in Flex über das Tag `<mx:DataGrid>` eingebunden und erhalten im Normalfall die Beschriftungen der Spalten automatisch.

Zur Navigation zwischen den einzelnen Monaten werden zwei Buttons erstellt, die die Navigation nach vorne und hinten ermöglichen sollen, wie Abbildung 7.64 zeigt.

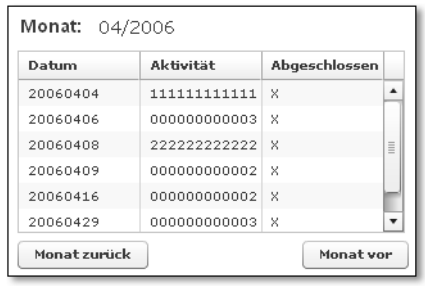


Abbildung 7.64 Rechte Seite der Disposition

Wie weiter oben angesprochen, werden die Elemente in einem `<mx:VBox>`-Tag gekapselt, um sie untereinander darzustellen. Allerdings befinden sich in diesem `VBox`-Element weitere `HBox`-Elemente zur horizontalen Darstellung des Monats und der beiden Buttons.

Darstellung des Monats

Zur Darstellung des Monats benötigen Sie zwei Labels in einem `<mx:HBox>`-Konstrukt: Das erste ist ein statischer Text, der den Text »Monat:« erhält, das zweite Label bekommt seinen Text dynamisch zugewiesen, was über ein Data-Binding an eine ActionScript-Variable geschieht:

```
<mx:HBox>
   <mx:Label text="Monat:" fontFamily="Arial"
      fontSize="14" fontWeight="bold"/>
   <mx:Label text="{l_month_displ}" fontSize="14"/>
</mx:HBox>
```

Neben der Definition des Textes weisen Sie beiden Labels die Schriftart, Größe und Darstellungsart (z. B. fett) zu. Das Data-Binding wird über die Variable `l_month_displ` erreicht, die Sie mittels geschweifter Klammern (`{}`) einbinden.

Einbindung des DataGrids

Wie bereits erwähnt, werden Tabellen über das Tag `<mx:DataGrid>` eingebunden. Dabei muss ein `dataProvider` angegeben werden, der die Daten enthält, die dargestellt werden sollen – in unserem Fall der Inhalt der XML-Datei `dispo_get.xml` vom SAP-Server.

Da die Spalten im `DataGrid` den Namen der XML-Tags erhalten[4], ist es sinnvoll, die Namen noch manuell zu korrigieren:

4 Das `<pernr>`-Tag würde im `DataGrid` den Spaltennamen **pernr** hervorrufen.

```
<mx:DataGrid sortableColumns="true" id="dg"
    dataProvider="">
   <mx:columns>
      <mx:DataGridColumn headerText="Datum"
          dataField="datum"/>
      <mx:DataGridColumn headerText="Aktivität"
          dataField="activity"/>
      <mx:DataGridColumn headerText="Abgeschlossen"
          dataField="finished"/>
   </mx:columns>
</mx:DataGrid>
```

Die Korrektur der Spaltennamen nehmen Sie über das Tag `<mx:columns>` vor, das die Kindelemente `<mx:DataGridColumn>` enthält. Die Tags erhalten die Eigenschaften `headerText` (Name der Spalte) und `dataField` (dargestelltes Element). Wir definieren unsere Überschriften als **Datum**, **Aktivität** und **Abgeschlossen**.

Vor-/Zurück-Buttons

Die beiden Buttons, die die Navigation zwischen den Monaten ermöglichen sollen, werden in einem `<mx:HBox>`-Element mit dem Zusatz des Attributs `horizontalGap` gekapselt, der angibt, welche Breite zwischen den Elementen der `HorizontalBox` freigelassen werden soll:

```
<mx:HBox horizontalGap="120">
   <mx:Button label="Monat zurück"
       click="button_click('zur',l_month)"/>
   <mx:Button label="Monat vor"
       click="button_click('vor',l_month)"/>
</mx:HBox>
```

Die beiden Buttons erhalten wiederum das Event `click` mit einer entsprechenden Funktion zugewiesen. Die Funktion `button_click` nimmt den aktuell in der Bearbeitung befindlichen Monat sowie die Richtung entgegen, in die navigiert werden soll.

Implementierung der Logik

Damit haben Sie die Layoutelemente der Applikation angebracht und können Sie nun mit Logik versehen. Um das Hinzufügen sowie das Auslesen und Darstellen von Dispositionen und Aktivitäten zu ermöglichen, müssen die `dataProvider` der entsprechenden Komponenten gefüllt werden. Dies erreichen Sie über ActionScript-Funkti-

onen (implementiert in `global.as` und `dispo.as`) und das Element `<mx:HTTPService>`.

Einbinden der XML-Dateien

Zum Einbinden einer XML-Datei, die auf einem Server liegt, wird das Tag `<mx:HTTPService>` benötigt. Da drei unterschiedliche XML-Dateien in der Disposition bearbeitet werden, müssen Sie entsprechend drei `<mx:HTTPService>`-Tags einbinden:

```
<!-- HTTPServices -->
<mx:HTTPService url="{l_url}" useProxy="false"
    id="dispo_data"/>
<mx:HTTPService url="{l_url_activity}" useProxy="false"
    id="activity_data"/>
<mx:HTTPService url="{l_url_send}" useProxy="false"
    id="dispo_add"
result="analyze_text(event.target.result.error_text)"/>
```

Ein `HTTPService` benötigt immer eine URL, an der sich die Datei befindet, die übermittelt werden soll. Diese URL wird dynamisch über ein Data-Binding (an den geschweiften Klammern erkennbar) gesetzt und aktualisiert. Das Ergebnis eines `HTTPService` wird dabei immer in die Variable `<id_des_services>.result` gestellt. Danach kann auf die einzelnen Elemente der XML-Datei einfach über `tag.tag.tag` zugegriffen werden.

Data-Binding

Hinweis

Bei einem Data-Binding wird der Attributwert automatisch geändert, sobald sich die im Data-Binding befindliche Variable ändert. Es muss also kein Event ausgelöst werden, um das Attribut den neuen Wert zu übermitteln.

Um die Daten des `HTTPService` später ansprechen zu können, wird ihnen eine ID zugeteilt. Weiterhin dient das Attribut `result` dazu, das Ergebnis vor der Ausgabe noch zu bearbeiten. Dazu muss eine Funktion hinterlegt sein, die den Inhalt der Ergebnis-XML-Datei entgegennimmt – in obigem Beispiel die Funktion `analyze_text`, die in der Datei `global.as` implementiert wird.

<mx:Script>

Die angesprochenen ActionScript-Dateien werden über das `<mx:Script>`-Tag mit dem `source`-Attribut in die Komponente eingebunden:

```
<mx:Script source="../as/global.as"/>
<mx:Script source="../as/dispo.as"/>
```

Da sich die MXML-Dateien im Verzeichnis `src` befinden, liegen die `as`-Dateien im relativen Pfad `../as/`. Der komplette Quelltext der Datei `dispo.mxml` ist in Listing 7.36 zu finden.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mx:Canvas xmlns:mx=http://www.adobe.com/2006/mxml
    xmlns="*" creationComplete="fire()" xmlns:comp="*">

   <mx:Script source="../as/global.as"/>
   <mx:Script source="../as/dispo.as"/>

   <mx:DateFormatter id="df_date"
       formatString="YYYYMMDD"/>

   <!-- HTTP-Services -->
   <mx:HTTPService url="{l_url}" useProxy="false"
       id="dispo_data"/>
   <mx:HTTPService url="{l_url_activity}"
      useProxy="false" id="activity_data"/>
   <mx:HTTPService url="{l_url_send}" useProxy="false"
       id="dispo_add"
       result=
       "analyze_text(event.target.result.error_text)"/>

   <mx:HBox>
      <!-- Linke Seite: Eingabe der Disposition -->
      <mx:VBox>
         <mx:Label text="Eingaben zur Disposition"
             fontSize="14" fontWeight="bold"
             fontFamily="Arial"/>
         <mx:Form id="form_dispo">
            <mx:FormItem label="Aktivität"
                required="true">
               <mx:ComboBox id="form_activity"
                  dataProvider=
                  "{activity_data.result.data.entry}"
                   labelField="value"/>
            </mx:FormItem>
            <mx:FormItem label="Datum" required="true">
               <mx:DateField id="form_date"
                   labelFunction="df_convert"/>
            </mx:FormItem>
            <mx:FormItem>
               <mx:Button id="form_submit"
                   label="Speichern"
```

```
                    click="add_disposition()"/>
            </mx:FormItem>
         </mx:Form>
      </mx:VBox>

      <!-- Rechte Seite: Anzeige der Dispositionen -->
      <mx:VBox>
         <mx:HBox>
            <mx:Label text="Monat:" fontFamily="Arial"
                fontSize="14" fontWeight="bold"/>
            <mx:Label text="{l_month_displ}"
                fontSize="14"/>
         </mx:HBox>

         <mx:DataGrid sortableColumns="true" id="dg"
            dataProvider=
            "{dispo_data.result.result.pos.entry}">
            <mx:columns>
               <mx:DataGridColumn headerText="Datum"
                   dataField="datum"/>
               <mx:DataGridColumn
                   headerText="Aktivität"
                   dataField="activity"/>
               <mx:DataGridColumn
                   headerText="Abgeschlossen"
                   dataField="finished"/>
            </mx:columns>
         </mx:DataGrid>

         <mx:HBox horizontalGap="120">
            <mx:Button label="Monat zurück"
                click="button_click('zur',l_month)"/>
            <mx:Button label="Monat vor"
                click="button_click('vor',l_month)"/>
         </mx:HBox>
      </mx:VBox>
   </mx:HBox>
</mx:Canvas>
```

Listing 7.36 Quelltext der Datei dispo.mxml

Erstellung der dataProvider

Wirft man einen Blick auf die `dataProvider` des `DataGrids` oder der `ComboBox`, so müssen diese nun über ActionScript gefüllt werden. Dazu ist es notwendig, die URLs der `<mx:HTTPService>`-Tags zu setzen, damit diese die Daten vom Backend-Server holen können.

Setzen des Monats

Die Applikation soll zu Beginn die Daten vom Server holen, die dem aktuellen Monat entsprechen. Dazu definieren Sie im ersten Schritt der Datei `dispo.as` das aktuelle Datum:

```
// Aktuelles Datum (für Startsicht benötigt)
private var l_month_actual:Date = new Date();
```

Dieses Datum wird so formatiert, dass es der Variablen der XML-Datei `dispo_get.xml` entspricht und entgegengenommen werden kann. Ist das aktuelle Datum beispielsweise »12.06.2006«, so muss es in »200606« umgewandelt werden:

```
// Aktueller Monat
[Bindable]
public var l_month:String =
   (++l_month_actual.month < 10) ?
   l_month_actual.fullYear.toString() + "0" +
   l_month_actual.month.toString() :
   l_month_actual.fullYear.toString() +
   l_month_actual.month.toString();
```

Die Anweisung ist ein Konstrukt folgender Art:

```
(A < B) ? X=10 : X=11;
```

Das bedeutet konkret: »Ist A kleiner als B, so setze X auf 10, ansonsten setze X auf 11.« Im Falle des Datums ist dieses Konstrukt notwendig, da das Flex-Datum die Eigenschaft hat, Monate nicht grundsätzlich zweistellig darzustellen – der Januar wird als 1 und nicht als 01 gekennzeichnet. Somit wird entschieden, ob der aktuelle Monat kleiner als 10 ist, falls ja, wird eine 0 vor dem Monat ergänzt, ansonsten kann der Monat so übernommen werden. Mit der Eigenschaft `fullYear` und `month` kann auf das Jahr respektive den Monat zugegriffen werden.

Monatsdarstellung

Für die Darstellung des Monats auf der rechten Seite der Disposition soll dieser das Format `MM/YYYY` erhalten. Diese Umwandlung nehmen Sie über einfache String-Konkatenationen vor:

```
// Aktueller Monat in darstellbarem Format
[Bindable]
public var l_month_displ:String = l_month.substr(4,2) +
   "/" + l_month.substr(0,4);
```

Darstellung der URL

Die URL muss alle entsprechenden Parameter enthalten, um die Datei `dispo_get.xml` korrekt ausführen zu können. Dazu müssen folgende Werte übergeben werden:

- Personalnummer (`pernr`)
- Monat (`month`)

Das Erstellen des URL-Strings `l_url` führen Sie wie folgt durch:

```
// Basis-URL und URL-Einstellungen
public var l_base_url:String =
   "http://server:port/sap/bc/bsp/sap/zdztb_bsp/";
   public var l_dataget:String = "data.xml";

// Endgültig aufzurufende URL
[Bindable]
public var l_url:String;

// URL der Aktivität
[Bindable]
public var l_url_activity:String = l_base_url +
   l_dataget + "?typ=A";

// Sendende URL
[Bindable]
public var l_url_send:String;
```

Dabei wird die endgültige URL über Hilfsvariablen definiert und konkateniert. Falls Ihnen die URL des Systems nicht bekannt ist, können Sie diese in Transaktion SE80 in der BSP-Applikation auslesen (siehe Abbildung 7.65). Dazu öffnen Sie Transaktion SE80 und wählen die BSP-Applikation ZDZTB_BSP und die darunter liegende Datei `dispo_add.xml` in den Seiten mit Ablauflogik. In den Eigenschaften der Datei finden Sie dann in der untersten Zeile die URL der Seite.

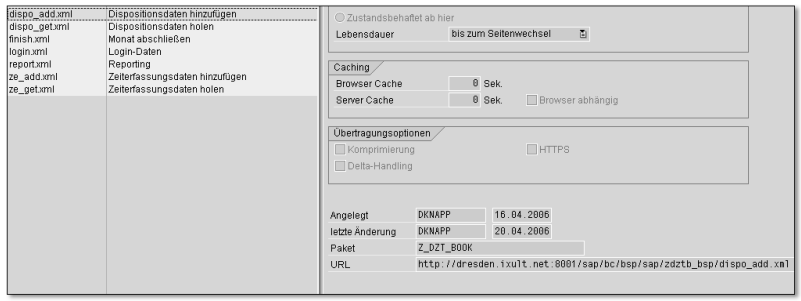


Abbildung 7.65 Auslesen des URL-Parameters

fire()

In der Datei `dispo.mxml` wird eine Funktion `fire()` im `<mx:Canvas>`-Tag im Event `CreationComplete` definiert. Dieses Event wird ausgelöst, sobald die Komponente geladen wurde und soll das Holen der XML-Dateien anstoßen. Da das reine Setzen der URL im `<mx:HTTPService>`-Tag nicht zum Holen der Daten ausreicht, muss dieses (beispielsweise) per ActionScript gelöst werden[5]; die Funktion `fire()` führt diese Aktion durch:

```
// Holen aller relevanten Daten
public function fire():void {
      dispo_data.send();
      activity_data.send();
}
```

Das Holen der Daten wird mit `<id_des_http_service>.send()` ausgelöst und wird in der Funktion `fire()` sowohl für die `ComboBox` (Aktivitäten) als auch das `DataGrid` (Dispositionsdaten) durchgeführt. Dadurch stehen sowohl dem `DataGrid` als auch der `ComboBox` die Daten zur Darstellung zur Verfügung, wie Abbildung 7.66 zeigt.

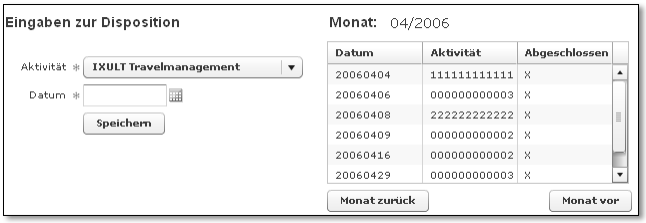


Abbildung 7.66 Aktivitäten und Dispositionen vom Server

Navigation zwischen den Monaten

Um zu einem Monat vor bzw. zurück zu navigieren, müssen Sie die Funktion `button_click(richtung,monat_aktuell)` implementieren, die in der Datei `dispo.mxml` an den Buttons hinterlegt wurde.

Dazu ist es notwendig, aus dem aktuellen Monat den neuen darzustellenden Monat zu berechnen. Dies realisieren Sie mit der Funktion `monat_action(richtung,monat_aktuell)`, die in der Datei `global.as` implementiert ist (siehe Listing 7.37).

```
// Einen Monat vor oder zurück navigieren
public function monat_action(richtung:String,
```

5 Dieses Vorgehen ist mit dem `onLoad`-Attribut eines `<body>`-HTML-Tags vergleichbar.

```
        monat_aktuell:String):String {

   var l_monat:Number;
   var l_monat_str:String;
   var l_jahr:Number;
   var l_jahr_str:String;
   var l_yearmon:String;

// Jahr und Monat holen
   l_jahr_str = monat_aktuell.substr(0,4);
   l_jahr = Number(l_jahr_str);

   l_monat_str = monat_aktuell.substr(4,2);
   l_monat = Number(l_monat_str);

// In Abhängigkeit der Richtung neuen Monat ermitteln
   if (richtung == "vor") {

      if (l_monat == 12) {
         l_monat = 1;
         l_jahr++;
      }
      else
         l_monat++;
   }
   else {

      if (l_monat == 1) {
         l_monat = 12;
         l_jahr--;
      }
      else
         l_monat--;
   }

// String auf zwei Zeichen ausdehnen
   if (l_monat < 10)
      l_monat_str = "0" + l_monat.toString();
   else
      l_monat_str = l_monat.toString();

   l_jahr_str = l_jahr.toString();
   l_yearmon = l_jahr_str + l_monat_str;

return l_yearmon;
}
```

Listing 7.37 Funktion monat_action(richtung,monat_aktuell)

Berechnung des Monats

Die Funktion berechnet in Abhängigkeit der Richtung und dem aktuellen Monat den neu darzustellenden Monat. Dazu muss im Normalfall der Monat um eins erhöht (Vorwärtsnavigation) bzw. erniedrigt (Rückwärtsnavigation) werden. Die beiden Spezialfälle Dezember (vorwärts) und Januar (rückwärts) werden getrennt behandelt, da hier eine Anpassung des Jahres notwendig ist.

Anschließend müssen Sie die Monatswerte wieder auf zwei Zeichen ausdehnen, um den String als `CHAR6`-Wert darzustellen. Mithilfe dieser Funktion ist es möglich, die Funktion `button_click(richtung,monat_aktuell)` zu implementieren (siehe Listing 7.38).

```
// Reaktion auf Vor- oder Zurück-Button (Disposition)
public function button_click(richtung:String,
                monat_aktuell:String):void {

   var l_yearmon:String;

// Neuen Monat holen
   l_yearmon = monat_action(richtung,monat_aktuell);

// Neue URL berechnen
   l_url = l_base_url + "dispo_get.xml?pernr=" +
           l_pernr + "&month=" + l_yearmon;
   l_month = l_yearmon;
   l_month_displ = l_month.substr(4,2) + "/" +
                   l_month.substr(0,4);

// Alte Sicht initialisieren, um im Falle keiner Daten
// nicht die alten Daten zu sehen
   dg.dataProvider = "";

// Abschicken des HTTP-Requests
   dispo_data.send();
}
```

Listing 7.38 Funktion button_click(richtung,monat_aktuell) im Dispositionsfall

Die Funktion holt sich mit der gerade definierten Funktion `monat_action` den neuen darzustellenden Monat. Anschließend wird die neue URL bestimmt (wodurch aufgrund des Data-Bindings automatisch die URL im `HTTPService` neu gesetzt wird) und die alte Sicht über `dg.dataProvider` initialisiert, um im Falle keiner neuen Daten nicht die alten Daten darzustellen. Dann ist ein Abschicken des HTTP-Requests über die `send()`-Funktion möglich.

Dadurch ist die Navigation zwischen den Monaten realisiert. Um den Rahmen der Beispielanwendung nicht zu sprengen, wird auf eine entsprechende Fehlerüberprüfung der Eingabe verzichtet.

Hinzufügen von Dispositionen

Das Hinzufügen einer Disposition realisieren Sie über die XML-Datei `dispo_add.xml`, der Sie Personalnummer, Datum und Aktivität übergeben. Diese werden in der Funktion `add_disposition()` aus den `Form`-Feldern der linken Seite der Dispositionsseite ausgelesen und daraus die sendende URL `l_url_send` erstellt (siehe Listing 7.39).

```
// Hinzufügen einer Disposition
public function add_disposition():void {

   var l_date:String = form_date.text;
   var l_activity:String =
       form_activity.selectedItem.name.toString();

// Sendende URL ermitteln
   l_url_send = l_base_url + "dispo_add.xml?pernr=" +
      l_pernr + "&datum=" + l_date + "&activity=" +
      l_activity;

// Abschicken der Requests (DISPO_ADD mit
// anschließendem DISPO_GET)
      dispo_add.send();
      dispo_data.send();
}
```

Listing 7.39 Funktion add_disposition()

Die Funktion ermittelt aus den beiden `Form`-Feldern `form_activity` und `form_date` die eingegebenen Texte und setzt damit die sendende URL zusammen. Anschließend werden über das `send()`-Kommando die HTTP-Requests abgeschickt, was das Senden der Daten mit anschließendem Holen der geänderten Daten bewirkt.

Fehlerausgabe

Sollten beim Hinzufügen der Disposition Fehler aufgetreten sein, werden diese automatisch über ein Alert-Fenster ausgegeben. Um dies zu erreichen, hinterlegen Sie die Funktion `analyze_text(error_text)` im `<mx:HTTPService>`-Tag der sendenden URL im `result`-Attribut. Die Funktion prüft, ob das Resultat fehlerfrei ist (der Wert **OK** wird zurückgegeben) oder nicht (Fehlertext vorhanden):

```
// Ausgabe von Fehlermeldungen
public function analyze_text(error:String):void {
   if (error != "OK")
      Alert.show(error,"Achtung!");
}
```

Diese Funktion ist ebenfalls Bestandteil der Datei `global.as`.

DateFormatter

Wie weiter oben in diesem Kapitel angesprochen, können Sie mit einer so genannten `labelFunction` ein Datum (oder anderes) vor der Darstellung so formatieren, wie man es in der Applikation benötigt. In unserem Fall wird das Datum im SAP-Format des Datentyps `DATS` benötigt (`YYYYMMDD`), sodass wir das vom `DateField` erzeugte Datum zuerst (automatisch) in dieses transformieren:

```
// Datumsformatierer
public function df_convert(date:Date):String {
   return df_date.format(date);
}

<mx:DateFormatter id="df_date"
    formatString="YYYYMMDD"/>

<mx:DateField id="form_date"
    labelFunction="df_convert"/>
```

Durch die Angabe von `df_convert(date)` wird bei der Ausgabe des Datums zuerst diese Funktion zur Manipulation des Datums verwendet. Diese Funktion ruft mit `df_date` den `DateFormatter` auf, der den neuen darzustellenden String (`formatString`) enthält. Die Konvertierung selbst wird dann automatisch von Flex übernommen.

Die Funktion `df_convert(date)` ist Bestandteil der ActionScript-Datei `global.as`, der `DateFormatter` und das `DateField` stammen aus `dispo.mxml`.

7.3.4 Zeiterfassungskomponente

Struktur der Zeiterfassung

Wir wollen die Zeiterfassung so gestalten, dass auf der linken Seite Zeiterfassungen vorgenommen werden können, die auf der rechten Seite dargestellt werden. Unser Ziel ist die Entwicklung der Zeiterfassungskomponente mit dem Layout aus Abbildung 7.67.

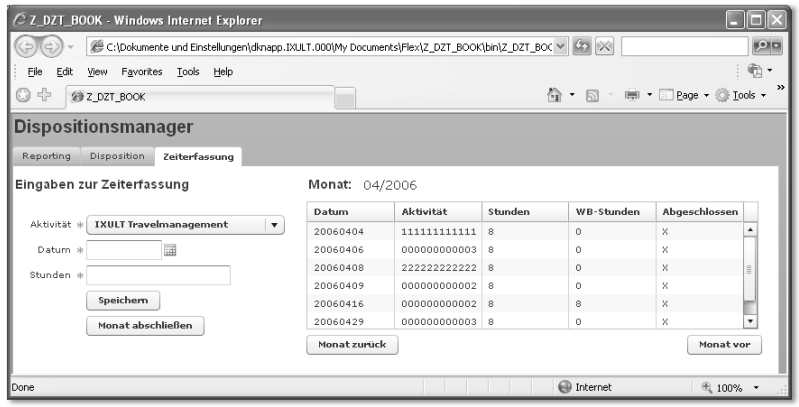


Abbildung 7.67 Darstellung der Zeiterfassungsseite

Wie man in Abbildung 7.67 sieht, ist die Zeiterfassung analog der Disposition in zwei Bereiche unterteilt: der linke zur Eingabe und der rechte zur Ausgabe. Auch diese beiden Bereiche müssen deshalb über ein `<mx:HBox>`-Tag voneinander getrennt werden:

```
<mx:HBox>
   <!-- Linke Seite -->
   <!-- Rechte Seite -->
</mx:HBox>
```

Notwendige Felder

Die zur Eingabe notwendigen Felder sind zum einen die Überschrift **Eingaben zur Zeiterfassung** sowie die Eingabefelder und die Buttons **Speichern** und **Monat abschließen**, die allesamt übereinander dargestellt werden müssen. Dazu ist wieder eine Aufgliederung der Elemente der linken Seite in ein `<mx:VBox>`-Tag notwendig. Das Gleiche gilt für die rechte Seite, die aus dem Monat, einer Tabelle und zwei Buttons besteht. Das folgende Listing 7.40 zeigt die Struktur mit diesen Layoutelementen.

```
<mx:HBox>
   <!-- Linke Seite: Eingabe der Zeiterfassung -->
   <mx:VBox>
      <!-- Eingabeelemente -->
   </mx:VBox>
```

```
   <!-- Rechte Seite: Anzeige der Zeiterfassungen -->
   <mx:VBox>
      <!-- Darstellungselemente -->
   </mx:VBox>
</mx:HBox>
```

Listing 7.40 Layoutelemente der Datei dispo.mxml

Linke Seite der Zeiterfassung

<mx:Form>

Nun können Sie die einzelnen Seiten mit funktionalen Elementen bestücken. Nach der Definition des Funktionsbausteins zum Hinzufügen (siehe Abschnitt 7.2.3) von Zeiterfassungen werden die Personalnummer, das Datum, die Stunden und die Aktivität benötigt. Die Personalnummer wird dabei beim Einloggen festgelegt, sodass der Anwender nur noch ein Datum, die Stunden und die entsprechende Aktivität auswählen muss. Wie auch bei der Disposition bedienen Sie sich hier des `<mx:Form>`-Tags:

- Das erste `FormItem`-Element dient zur Darstellung der Aktivität und enthält das entsprechende Label. Die Eigenschaft `required` gibt an, ob das Item ein Mussfeld ist oder nicht.
- Das `FormItem` selbst enthält die `ComboBox`, die die Aktivitäten darstellen soll. Die Aktivitäten stammen dabei aus dem Frontend und werden über die Datei `data.xml` an das Frontend übermittelt. Das Füllen der `ComboBox` übernimmt die Eigenschaft `dataProvider`, auf die im Laufe des Kapitels noch eingegangen wird.

  Wichtig ist, dass Sie die Eigenschaft `labelField` setzen, um zu kennzeichnen, welches Element der XML-Struktur später dargestellt werden soll.
- Das nächste `FormItem` dient zur Erfassung der geleisteten Stunden. Hierzu nutzen Sie das `<mx:TextInput>`-Tag. Das `TextInput`-Feld ist zu vergleichen mit dem HTML-Tag `<input>` und hat die gleichen Eigenschaften.
- Das zweite `FormItem`-Element stellt das Datum dar. Bestücken Sie es ebenfalls mit der Eigenschaft `required`. Das `FormItem` enthält als Kind das Tag `<mx:DateField>`, dem neben der ID die `labelFunction` zugewiesen wird (analog zur Disposition), die wieder die Funktion `df_convert(date)` erhält.

- Um die getätigten Eingaben an das Backend zu übermitteln, wird ein Button benötigt. Dieser Button wird über das vierte `FormItem` eingebunden und enthält das Tag `<mx:Button>`. Diesem Tag weisen Sie wieder das System-Event `click` zu, das mit der Funktion `add_ze()` bestückt ist.
- Neben der Funktion der Übermittlung von Zeiterfassungen kann ein Monat auch abgeschlossen werden, d.h. es können keine Erfassungen mehr für diesen Zeitraum vorgenommen werden. Auch hier definieren Sie ein `<mx:Button>`-Tag mit dem Standard-Event `click`, das von der Funktion `month_complete()` bearbeitet wird.

Das `<mx:Form>`-Tag der Zeiterfassung ist in Listing 7.41 gezeigt.

```
<mx:Form id="form_ze">
   <mx:FormItem label="Aktivität" required="true">
      <mx:ComboBox id="form_activity"
          dataProvider="" labelField="value"/>
   </mx:FormItem>
   <mx:FormItem label="Datum" required="true">
      <mx:DateField id="form_date"
          labelFunction="df_convert"/>
   </mx:FormItem>
   <mx:FormItem label="Stunden" required="true">
      <mx:TextInput id="form_std"/>
   </mx:FormItem>
   <mx:FormItem>
      <mx:Button id="form_submit" label="Speichern"
          click="add_ze()"/>
   </mx:FormItem>
   <mx:FormItem>
      <mx:Button id="form_complete" label="Monat
          abschließen" click="month_complete()"/>
   </mx:FormItem>
</mx:Form>
```

Listing 7.41 Form-Elemente der Disposition

Das Layout in Abbildung 7.68 ist das Ergebnis von Listing 7.41.

Abbildung 7.68 Layout des <mx:Form>-Tags

Rechte Seite der Zeiterfassung

Die rechte Seite der Zeiterfassung ist weitestgehend analog zur rechten Seite der Disposition zu sehen. Aus diesem Grunde wird hier nicht genauer darauf eingegangen, es gelten die Informationen aus der rechten Seite der Disposition.

Einbinden der XML-Dateien

Die entsprechenden XML-Dateien werden wieder über das `<mx:HTTPService>`-Tag realisiert. Da drei unterschiedliche XML-Dateien in der Zeiterfassung bearbeitet werden, müssen Sie drei Tags einbinden:

```
<mx:HTTPService url="{l_url}" useProxy="false"
    id="ze_data"/>
<mx:HTTPService url="{l_url_activity}" useProxy="false"
    id="activity_data"/>
<mx:HTTPService url="{l_url_send}" useProxy="false"
    id="ze_add"
    result="analyze_text(event.target.result.error_text)"/>
```

Um die Daten des `HTTPService` später ansprechen zu können, wird ihnen jeweils eine ID zugeteilt. Weiterhin dient das Attribut `result` dazu, das Ergebnis vor der Ausgabe noch zu bearbeiten. Dazu muss eine Funktion hinterlegt sein, die den Inhalt der Ergebnis-XML-Datei entgegennimmt – in obigem Beispiel die Funktion `analyze_text`, die in der Datei `global.as` implementiert wird.

<mx:Script>

Binden Sie die angesprochenen ActionScript-Dateien über das `<mx:Script>`-Tag mit dem Attribut `source` in die Komponente ein:

```
<mx:Script source="../as/global.as"/>
<mx:Script source="../as/ze.as"/>
```

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z