ting 9.27 gezeigt, im Top-Include der Funktionsgruppe deklariert haben. Die Funktionsgruppe benötigt keine globalen Daten.

screen_handler

Listing 9.27 Deklaration der Klasse screen_handler im Top-Include

```
FUNCTION-POOL z_airlines.

CLASS screen_handler DEFINITION CREATE PRIVATE.
  PUBLIC SECTION.
    CLASS-METHODS create_screen.
    METHODS constructor.
  PRIVATE SECTION.
    CLASS-DATA screen TYPE REF TO screen_handler.
    DATA: container_html
            TYPE REF TO cl_gui_custom_container,
          container_box
            TYPE REF TO cl_gui_dialogbox_container,
          picture     TYPE REF TO cl_gui_picture,
          tree        TYPE REF TO cl_gui_simple_tree,
          html_viewer TYPE REF TO cl_gui_html_viewer,
          alv         TYPE REF TO cl_salv_table.
    METHODS: fill_tree,
             fill_picture,
             fill_html
               IMPORTING carrid TYPE spfli-carrid,
             fill_list
               IMPORTING carrid TYPE spfli-carrid
                         connid TYPE spfli-connid,
             handle_node_double_click
               FOR EVENT node_double_click
               OF cl_gui_simple_tree
               IMPORTING node_key,
             close_box
               FOR EVENT close
               OF cl_gui_dialogbox_container.
ENDCLASS.
```

Die öffentliche statische Methode `create_screen` dient der Instanzierung der nur privat instanzierbaren Klasse. Sämtliche Instanzattribute außer `alv` sind Referenzvariablen, die sich auf Klassen des CFW beziehen. Sie erkennen zwei Klassen für Container-Controls und drei Klassen für Anwendungs-Controls. Der statische Typ der Referenzvariablen `alv` ist die Verschalungsklasse CL_SALV_GRID für das ALV Grid Control. Wir haben vier Instanzmethoden zum Füllen der Anwendungs-Controls und zwei Ereignisbehandlermethoden zur Reaktion auf Benutzeraktionen deklariert.

Der Implementierungsteil der Klasse befindet sich im Include-Programm LZ_AIRLINESP01. Wir gehen im Folgenden auf die Implementierung jeder Methode ein.

create_screen

Listing 9.28 zeigt die Implementierung der statischen Methode `create_screen`. Sie erzeugt mit dem statischen Attribut `screen` ein Objekt der Klasse `screen_handler`. Dabei wird der Instanzkonstruktor `constructor` ausgeführt.

Listing 9.28 Methode create_screen

```
METHOD create_screen.
  IF screen IS INITIAL.
    CREATE OBJECT screen.
  ENDIF.
ENDMETHOD.
```

constructor

Listing 9.29 zeigt den Instanzkonstruktor. Als Erstes erzeugen wir ein Docking Control mit der lokalen Referenzvariablen `docking`. Durch Übergabe der Konstanten `dock_at_left` der Klasse CL_GUI_DOCKING_CONTAINER an den Parameter `side` wird ein neuer Bereich mit einer Breite von 135 Pixeln (Parameter `extension`) auf der linken Bildschirmseite erzeugt.

Listing 9.29 Instanzkonstruktor

```
METHOD constructor.
  DATA: event_tab TYPE cntl_simple_events,
        event     LIKE LINE OF event_tab,
        docking   TYPE REF TO cl_gui_docking_container,
        splitter  TYPE REF TO cl_gui_splitter_container,
        container_top
                  TYPE REF TO cl_gui_container,
        container_bottom
                  TYPE REF TO cl_gui_container.

  CREATE OBJECT container_html
    EXPORTING container_name = 'CUSTOM_CONTROL'.

  CREATE OBJECT docking
    EXPORTING side  = cl_gui_docking_container=>dock_at_left
              extension = 135.

  CREATE OBJECT splitter
    EXPORTING parent  = docking
```

```
                rows    = 2
                columns = 1.

  splitter->set_border( border = cl_gui_cfw=>false ).
  splitter->set_row_mode( mode = splitter->mode_absolute ).
  splitter->set_row_height( id  = 1 height = 180 ).

  container_top =
    splitter->get_container( row = 1 column = 1 ).
  container_bottom =
    splitter->get_container( row = 2 column = 1 ).

  CREATE OBJECT picture
    EXPORTING parent = container_top.

  CREATE OBJECT tree
    EXPORTING parent = container_bottom
              node_selection_mode =
                cl_gui_simple_tree=>node_sel_mode_single.

  event-eventid =
    cl_gui_simple_tree=>eventid_node_double_click.
  event-appl_event = ' '.
  APPEND event TO event_tab.
  tree->set_registered_events( events = event_tab ).
  SET HANDLER me->handle_node_double_click FOR tree.

  me->fill_picture( ).
  me->fill_tree( ).
ENDMETHOD.
```

Diesen Bildschirmbereich teilen wir in zwei horizontale Bereiche auf, indem wir über die lokale Referenzvariable `splitter` ein Objekt der Klasse CL_GUI_SPLITTER_CONTAINER erzeugen und dem Parameter `parent` die Referenz auf das Docking Control übergeben. Die Aufteilung erfolgt über die Parameter `rows` und `columns`. Durch Aufruf von Methoden des Splitter Controls legen wir fest, dass kein Rahmen gezeichnet und dass die Höhe absolut angegeben wird, wofür wir oben 180 Pixel angeben.

Mit der funktionalen Methode `get_container` lesen wir die Referenzen auf die beiden Bereiche des Splitter Controls in die lokalen Referenzvariablen `container_top` und `container_bottom` ein. Danach erzeugen wir ein Picture Control und ein Tree Control über die

Instanzattribute `picture` und `tree`, wobei wir dem Parameter `parent` jedes Konstruktors jeweils eine der Referenzen auf die Bereiche des Splitter Controls übergeben.

Damit unsere Ereignisbehandlermethode `handle_node_double_click` auf Doppelklicks auf Knoten des Tree Controls reagieren kann, müssen wir dieses Ereignis aktivieren und den Ereignisbehandler registrieren. Für die Aktivierung wird die Methode `set_registered_events` aufgerufen, die in jedem Anwendungs-Control zur Verfügung steht. Als Aktualparameter dient die interne Tabelle `event_tab`, die wir lokal mit dem Typ `cntl_simple_events` deklariert haben. In der Spalte `eventid` spezifizieren wir das Doppelklick-Ereignis. Dadurch, dass wir die Spalte `appl_event` nicht mit einem »X« füllen, definieren wir, dass das Ereignis als Systemereignis und nicht als Anwendungsereignis behandelt wird. Danach registrieren wir den Ereignisbehandler.

Nach Aufruf der Methoden `fill_picture` und `fill_tree` wird der Konstruktor verlassen, wodurch sämtliche lokalen Referenzvariablen des Konstruktors gelöscht werden. Die im Konstruktor mit lokalen Referenzvariablen erzeugten Control-Objekte werden dennoch nicht vom Garbage Collector aufgesammelt, da sie von Referenzen aus dem CFW am Leben gehalten werden.

fill_picture

Listing 9.30 zeigt die Methode `fill_picture`. Da es keine angemessene Verschalung für dieses GUI Control gibt,[20] haben wir mit der Klasse ZCL_PICTURE eine Hilfsklasse angelegt, die wenigstens die technische Vorbereitung einer Bilddatei für das Picture Control auslagert (wir beschreiben die Klasse am Endes dieses Abschnitts). Die Methode FACTORY der Klasse ZCL_PICTURE erwartet die Adresse einer Bilddatei im MIME Repository und liefert eine Referenz auf ein Objekt zurück, das die Darstellung des Bildes in einem Picture Control ermöglicht. Zur Darstellung dient die Methode DISPLAY, an die eine Referenz auf ein Picture Control übergeben wird.

Listing 9.30 Methode fill_picture

```
METHOD fill_picture.
    DATA pict TYPE REF TO zcl_picture.
    TRY.
        pict = zcl_picture=>factory(
```

20 Zumindest ist uns keine bekannt.

```
        mime_file =
          '/SAP/PUBLIC/BC/ABAP/Sources/PLANE.GIF'
        pict_type = 'GIF' ).
    CATCH zcx_no_picture.
      RETURN.
  ENDTRY.
  pict->display( picture ).
ENDMETHOD.
```

MIME Repository

Das MIME Repository ist ein Teil des Repository auf der Datenbank eines AS ABAP, in dem Multimedia-Dateien wie in einem Dateisystem gespeichert werden können (der Begriff MIME geht auf *Multipurpose Internet Mail Extension*, ein Verfahren zum Verpacken von Multimedia-Inhalten in SMTP-Mails, zurück). Diese Dateien sind wie alle Repository-Objekte an das Transportwesen angeschlossen. Sie können das MIME Repository im Object Navigator über die Drucktaste **MIME Repository** aufrufen, um dort Dateien abzulegen und zu verwalten. Wir verwenden das Bild eines Flugzeugs im GIF-Format, das wir für diesen Zweck in das MIME Repository unserer Trial Version des AS ABAP geladen haben.[21] Die Browser-basierten Oberflächen BSP und Web Dynpro haben übrigens einen viel direkteren Zugang zum MIME Repository als die klassischen Dynpros.

fill_tree

Listing 9.31 zeigt die Methode `fill_tree`. Die Methode `fill_tree` erzeugt die Baumstruktur des Tree Controls. Hierfür müssen wir eine interne Tabelle vorgegebener Struktur füllen. Wir verwenden zu diesem Zweck die vorgegebene Struktur MTREESNODE aus dem ABAP-Dictionary. Falls diese Struktur für Ihre Zwecke nicht ausreicht, können Sie auch selbst definierte Strukturen verwenden, die am Anfang TREEV_NODE einbinden.

Listing 9.31 Methode fill_tree

```
METHOD fill_tree.

  DATA: node_table TYPE TABLE OF mtreesnode,
        node       TYPE mtreesnode,
        spfli      TYPE spfli,
        spfli_tab  TYPE zcl_airlines=>tspfli_tab.

  spfli_tab = zcl_airlines=>get_spfli_tab( ).

  node-isfolder = 'X'.
  LOOP AT spfli_tab INTO spfli.
    AT NEW carrid.
      node-node_key = spfli-carrid.
      CLEAR node-relatkey.
      CLEAR node-relatship.
      node-text = spfli-carrid.
      CLEAR node-n_image.
      CLEAR node-exp_image.
      APPEND node TO node_table.
    ENDAT.
    AT NEW connid.
      CONCATENATE spfli-carrid spfli-connid
        INTO node-node_key.
      node-relatkey = spfli-carrid.
      node-relatship = cl_gui_simple_tree=>relat_last_child.
      node-text = spfli-connid.
      CALL FUNCTION 'ICON_CREATE'
        EXPORTING
          name   = 'ICON_FLIGHT'
          info   = text-con
        IMPORTING
          result = node-n_image
        EXCEPTIONS
          others = 4.
      IF sy-subrc <> 0.
        node-n_image = '@AV@'.
      ENDIF.
      node-exp_image = node-n_image.
    ENDAT.
    APPEND node TO node_table.
  ENDLOOP.

  tree->add_nodes( table_structure_name = 'MTREESNODE'
                   node_table           = node_table ).

ENDMETHOD.
```

Wir haben eine interne Tabelle `node_table` mit der Zeilenstruktur von MTREESNODE angelegt. Jede Zeile der Tabelle beschreibt einen Knoten der Baumstruktur und muss dafür einen eindeutigen Kno-

21 Ab dem nächsten Release (SAP NetWeaver 2007) ist das Bild standardmäßig unter diesem Pfad vorhanden, da es dann auch von Beispielen der ABAP-Dokumentation verwendet wird.

tenschlüssel node_key haben.[22] Die Komponenten relatkey und relatship beschreiben die Beziehungen zwischen den Knoten. Andere Komponenten erlauben es, die Standardicons zu ändern oder Knotentexte anzulegen etc.

In unserem Beispiel erzeugen wir die Knotentabelle aus dem Inhalt einer internen Tabelle spfli_tab, die wir gemäß der »Separation of Concerns« natürlich nicht direkt in der Funktionsgruppe, sondern in einer Serviceklasse ZCL_AIRLINES mit Daten aus der Datenbanktabelle SPFLI füllen. Mit Hilfe der Gruppenstufenverarbeitung AT – ENDAT füllen wir die Knotentabelle in einer LOOP-Schleife mit Zeilen, die eine zweistufige Hierarchie darstellen. Vor der LOOP-Schleife werden einige allgemeine Knoteneigenschaften für alle Zeilen in den Arbeitsbereich node gestellt.

CREATE_ICON

Bei den Unterknoten ersetzen wir die Standardicons für nicht expandierte und expandierte Knoten durch Flugzeugicons, indem wir den Komponenten n_image und exp_image die interne Darstellung für diese Icons zuweisen, die wir mit dem Standard-Funktionsbaustein CREATE_ICON erzeugen. Dieser Funktionsbaustein ist allgemein für die Erstellung von Icons auf Dynpros verwendbar und erlaubt es, zusätzliche Texte und Quickinfo-Texte anzugeben. Letztere sind für barrierefreie Programme sogar vorgeschrieben. Sie finden die benötigten Iconnamen durch Ausführen des Programms SHOWICON. Als Ausnahmebehandlung geben wir die interne Kennung des Icons »@AV@« ohne Quickinfo-Text an.

fill_html

Listing 9.32 zeigt die Methode fill_html. Die Methode fill_html erzeugt bei ihrem ersten Aufruf ein Objekt der Klasse CL_GUI_HTML_VIEWER und verbindet das zugehörige HTML Control mit dem Bereich des Container-Controls, auf das die Referenz in container_html zeigt, nämlich mit dem Custom-Control auf dem Dynpro. Beachten Sie, dass ohne die Anweisung IF bei jedem erneuten Aufruf der Methode ein weiteres HTML Control erzeugt würde. Alle diese Controls wären aktiv im ABAP-Programm und auf dem Präsentationsserver vorhanden, wenn auch nur das jeweils letzte sichtbar ist. Sie können also durchaus mehrere Anwendungs-Controls mit einem einzigen Container-Control verbinden. Die wechselseitige Sichtbarkeit können Sie dann mit der Methode SET_VISIBLE steuern.

22 Dieser Schlüssel ist auf zwölf Zeichen begrenzt. Es gibt aber auch eine Klasse CL_SIMPLE_TREE_MODEL, die CL_GUI_SIMPLE_TREE verschalt und diese Einschränkung aufhebt. Für unser Beispiel hätte die Verwendung der Verschalung keine Vorteile.

Listing 9.32 Methode fill_html

```
METHOD fill_html.
  DATA url TYPE zcl_airlines=>turl.
  IF html_viewer IS INITIAL.
    CREATE OBJECT html_viewer
           EXPORTING parent = container_html.
  ENDIF.
  url = zcl_airlines=>get_url( carrid ).
  html_viewer->show_url( url = url ).
ENDMETHOD.
```

Über die Serviceklasse ZCL_AIRLINES besorgen wir uns die URL-Adresse einer Fluggesellschaft und übergeben sie der Methode show_url unseres HTML Controls, das dadurch die Homepage der Fluggesellschaft darstellt.

fill_list

Listing 9.33 zeigt die Methode fill_list. Die Methode fill_list erzeugt einen Dialogbox-Container, falls nicht bereits einer vorhanden ist. Für den Dialogbox-Container wird der Ereignisbehandler close_box registriert. Eine Aktivierung mit der Methode SET_REGISTERED_EVENTS ist hier nicht notwendig, da das Ereignis CLOSE der Klasse CL_GUI_DIALOGBOX_CONTAINER in der Klasse selbst als Systemereignis aktiviert wird. Wir haben den Dialogbox-Container hier zur Demonstration selbst programmiert. In ihm können beliebige GUI Controls angezeigt werden. Wenn er wie in unserem Beispiel nur zur Anzeige einer ALV-Liste benötigt wird, könnte hierfür auch die Methode SET_SCREEN_POPUP der ALV-Klasse verwendet werden (siehe Beispiel in Listing 9.51).

Listing 9.33 Methode fill_list

```
METHOD fill_list.

  DATA flight_tab TYPE zcl_airlines=>tflight_tab.
  DATA settings   TYPE REF TO cl_salv_display_settings.
  DATA list_title TYPE lvc_title.

  IF container_box IS INITIAL.
    CREATE OBJECT container_box
```

```
          EXPORTING width  = 300
                    height = 200
                    top    = 100
                    left   = 400
                    caption = text-fli.
  SET HANDLER close_box FOR container_box.
ENDIF.

flight_tab = zcl_airlines=>get_flight_tab(
               carrid = carrid
               connid = connid ).

CONCATENATE carrid ` ` connid INTO list_title.

TRY.
    IF alv IS INITIAL.
      cl_salv_table=>factory(
        EXPORTING r_container  = container_box
        IMPORTING r_salv_table = alv
        CHANGING  t_table      = flight_tab ).
      alv->display( ).
    ELSE.
      alv->set_data( CHANGING t_table = flight_tab ).
    ENDIF.
    settings = alv->get_display_settings( ).
    settings->set_list_header( list_title ).
    alv->refresh( ).
  CATCH cx_salv_msg cx_salv_no_new_data_allowed.
    MESSAGE text-alv TYPE 'I' DISPLAY LIKE 'E'.
ENDTRY.

ENDMETHOD.
```

CL_SALV_TABLE

Über die Serviceklasse ZCL_AIRLINES lesen wir ein paar Detaildaten zu einem ausgewählten Knoten in eine interne Tabelle `flight_tab` und erzeugen einen Titel für eine ALV-Liste. Falls noch kein Objekt für die ALV-Liste vorhanden ist, erzeugen wir dies und übergeben ihm dabei die interne Tabelle mit den Daten. Durch Zuweisung der Referenz auf den Dialogbox-Container wird die ALV-Liste an diesen gebunden und durch Aufruf ihrer Methode DISPLAY dargestellt. Ohne Verwendung des `EXPORTING`-Zusatzes bei `CREATE OBJECT` würde die ALV-Liste in einem eigenen Dynpro im gesamten aktuellen Fenster dargestellt. Falls die ALV-Liste schon vorhanden ist, werden nur neue Daten übergeben. Der Titel der ALV-Liste wird über ein von `settings` referenziertes Einstellungsobjekt gesetzt. Der Aufruf der Methode REFRESH übermittelt den Titel und auch neu übergebene Daten an die Anzeige. Wenn Ihnen die letzte Auflage dieses Buches vorliegt, vergleichen Sie die damalige Verwendung des ALV Grid Controls mit der hier gezeigten Verwendung der Verschalung durch CL_SALV_TABLE. Sie sehen, dass wir jetzt von den Interna der Listendarstellung weitestgehend befreit sind.

handle_node_double_click

Listing 9.34 zeigt die Methode `handle_node_double_click`. Die Methode `handle_node_double_click` ist als Ereignisbehandler für Doppelklicks auf Knoten der Baumstruktur deklariert und wurde im Konstruktor dafür registriert. Sie wird also durch eine solche Benutzeraktion aufgerufen. Die Methode importiert den Parameter `node_key` des Ereignisses `node_double_click` der Klasse CL_GUI_SIMPLE_TREE, der den Schlüssel des ausgewählten Knotens enthält. Wir teilen den Schlüssel in die Teile auf, aus denen wir ihn in der Methode `fill_tree` zusammengesetzt haben, und rufen je nach Inhalt eine der Methoden `fill_html` oder `fill_list` auf.

Listing 9.34 Methode handle_node_double_click

```
METHOD handle_node_double_click.

  DATA: carrid TYPE spfli-carrid,
        connid TYPE spfli-connid.

  carrid = node_key(2).
  connid = node_key+2(4).
  IF connid IS INITIAL.
    fill_html( carrid = carrid ).
  ELSE.
    fill_list( carrid = carrid
               connid = connid ).
  ENDIF.

ENDMETHOD.
```

Synchronisationspunkt

Die Methodenaufrufe werden in der Automation Queue bis zum nächsten Synchronisationspunkt zwischen Applikations- und Präsentationsserver gespeichert. Nach der Behandlung von Systemereignissen enthält das CFW einen automatischen Synchronisationspunkt, wodurch die Methoden noch während der Ereignisbehandlung ausgeführt werden.

close_box Listing 9.35 zeigt die Methode close_box. Die Methode close_box gibt dem Benutzer die Möglichkeit, die Dialogbox mit dem üblichen Icon zu schließen. Hierfür werden nacheinander die Methoden close_screen und free von der ALV-Liste und dem Dialogbox-Container aufgerufen und dann die zugehörigen Referenzvariablen initialisiert.

Listing 9.35 Methode close_box

```
METHOD close_box.
    alv->close_screen( ).
    container_box->free( ).
    CLEAR: alv, container_box.
ENDMETHOD.
```

ZCL_PICTURE Listing 9.36 zeigt die Klasse ZCL_PICTURE, die wir zur Verschalung von Bildern für das Picture Control eingeführt haben. Die Methode FACTORY erzeugt ein Objekt, das als Attribute eine interne Tabelle mit einer Bilddatei und eine URL für diese Tabelle enthält.

Listing 9.36 Klasse ZCL_PICTURE

```
CLASS zcl_picture DEFINITION PUBLIC
  FINAL CREATE PRIVATE.
  PUBLIC SECTION.
    TYPES: tpict_line TYPE x LENGTH 1022,
           tpict_tab  TYPE TABLE OF tpict_line.
    METHODS constructor
      IMPORTING
        mime_file TYPE csequence
        pict_type TYPE c
      RAISING
        zcx_no_picture.
    CLASS-METHODS factory
      IMPORTING
        mime_file TYPE csequence
        pict_type TYPE c
      RETURNING
        value(picture) TYPE REF TO zcl_picture
      RAISING
        zcx_no_picture.
    METHODS display
      IMPORTING
        picture_control TYPE REF TO cl_gui_picture.
  PRIVATE SECTION.
    DATA pict_tab TYPE tpict_tab.
    DATA url TYPE char255.
ENDCLASS.

CLASS zcl_picture IMPLEMENTATION.

  METHOD constructor.
    DATA  mime_api   TYPE REF TO if_mr_api.
    DATA  pict_wa    TYPE xstring.
    DATA  strl       TYPE i.

    mime_api = cl_mime_repository_api=>get_api( ).
    mime_api->get(
      EXPORTING i_url = mime_file
      IMPORTING e_content = pict_wa
      EXCEPTIONS OTHERS = 4 ).
    IF sy-subrc <> 0.
      RAISE EXCEPTION TYPE zcx_no_picture.
    ENDIF.

    strl = xstrlen( pict_wa ).
    WHILE strl >= 1022.
      APPEND pict_wa(1022) TO pict_tab.
      SHIFT pict_wa BY 1022 PLACES LEFT IN BYTE MODE.
      strl = xstrlen( pict_wa ).
    ENDWHILE.
    IF strl > 0.
      APPEND pict_wa TO pict_tab.
    ENDIF.

    CALL FUNCTION 'DP_CREATE_URL'
      EXPORTING
        type    = 'IMAGE'
        subtype = pict_type
      TABLES
        data    = pict_tab
      CHANGING
        url     = url.
  ENDMETHOD.

  METHOD factory.
    CREATE OBJECT picture
      EXPORTING mime_file = mime_file
                pict_type = pict_type.
  ENDMETHOD.
```

```
  METHOD display.
    picture_control->load_picture_from_url( url = url ).
    picture_control->set_display_mode( display_mode =
      picture_control->display_mode_stretch ).
  ENDMETHOD.

ENDCLASS.
```

Im Instanzkonstruktor wird das angeforderte MIME Repository über das zugehörige API ausgelesen.[23] Da das Ergebnis ein Bytestring ist, das Picture Control aber nur mit einer internen Tabelle arbeitet, müssen wir den Inhalt des Bytestring `pict_wa` in die interne Tabelle `pict_tab` einlesen. Für das Setzen der URL auf das Bild in der Tabelle wird der Funktionsbaustein DP_CREATE_URL verwendet.

Die Methode DISPLAY übergibt die URL an die Methode `load_picture_from_url` des Picture Controls und sendet dadurch das Bild in den Bereich des Controls auf dem Bildschirm. Mit der Methode `set_display_mode` legen wir fest, dass das Bild sich immer der aktuellen Größe des Bereichs anpasst.

ZCL_AIRLINES

Listing 9.37 zeigt der Vollständigkeit halber noch die Klasse ZCL_AIRLINES, die wir zur Erzeugung der Daten des Beispiels angelegt haben. Jede Methode enthält eine passende `SELECT`-Anweisung. Beachten Sie, dass ein Verwender der Klasse nichts von den Datenbanktabellen wissen muss. Ihm genügen die Datentypen, die im öffentlichen Sichtbarkeitsbereich der Klasse deklariert sind.

Listing 9.37 Klasse ZCL_AIRLINES

```
CLASS zcl_airlines DEFINITION PUBLIC
  FINAL CREATE PUBLIC.
  PUBLIC SECTION.
    TYPES:
      tspfli_tab TYPE SORTED TABLE OF spfli
                 WITH UNIQUE KEY carrid connid,
      turl TYPE scarr-url,
      tflight_tab TYPE STANDARD TABLE OF demofli
                  WITH NON-UNIQUE DEFAULT KEY.
    CLASS-METHODS get_spfli_tab
      RETURNING
        value(spfli_tab) TYPE tspfli_tab.
    CLASS-METHODS get_url
      IMPORTING
        carrid TYPE scarr-carrid
      RETURNING
        value(url) TYPE turl.
    CLASS-METHODS get_flight_tab
      IMPORTING
        carrid TYPE sflight-carrid
        connid TYPE sflight-connid
      RETURNING
        value(flight_tab) TYPE tflight_tab.
ENDCLASS.

CLASS zcl_airlines IMPLEMENTATION.

  METHOD get_spfli_tab.
    SELECT carrid connid
           FROM spfli
           INTO CORRESPONDING FIELDS OF TABLE spfli_tab.
  ENDMETHOD.

  METHOD get_url.
    SELECT SINGLE url
           FROM   scarr
           INTO   url
           WHERE  carrid = carrid.
  ENDMETHOD.

  METHOD get_flight_tab.
    SELECT fldate seatsmax seatsocc
           INTO CORRESPONDING FIELDS OF TABLE flight_tab
           FROM sflight
           WHERE carrid = carrid AND connid = connid
           ORDER BY fldate.
  ENDMETHOD.

ENDCLASS.
```

Zusammen

Die Vorgehensweise unseres Beispiels zu GUI Controls lässt sich in folgenden Schritten zusammenfassen, die allgemein gültig für den Umgang mit Controls sind:

1. Erzeugung von Container-Controls und Anbindung an Bildschirmbereiche von Dynpros

23 Wenn Sie die letzte Auflage dieses Buches kennen, werden Sie sich daran erinnern, dass wir uns damals noch – horribile dictu – eine Art eigene MIME-Ablage für die Bilddatei gebastelt haben. Vielleicht ist ja in fünf Jahren auch keine eigene Verschalung für das Picture Control mehr notwendig.

2. Erzeugung von Anwendungs-Controls oder Objekten von Verschalungsklassen und Anbindung an die Container-Controls
3. Bereitstellung von Daten, um die Controls zu füllen, in den meisten Fällen in internen Tabellen von speziellem Typ
4. Senden der Daten an die Anwendungs-Controls oder Verschalungsobjekte
5. Reaktion auf Benutzerereignisse durch die Implementierung und Registrierung geeigneter Behandlermethoden

9.2 Selektionsbilder

Selektionsbilder sind spezielle Dynpros, die mit ABAP-Anweisungen definiert werden, ohne dass der Screen Painter verwendet werden muss. Bei der Aktivierung eines ABAP-Programms mit Selektionsbildern generiert die ABAP-Laufzeitumgebung diese Dynpros mit allen ihren Bestandteilen inklusive der Ablauflogik. Im ABAP-Programm müssen keine Dialogmodule für Selektionsbilder angelegt werden. Stattdessen führen Benutzeraktionen auf Selektionsbildern zu speziellen Selektionsbildereignissen, die in Ereignisblöcken behandelt werden können.

Ursprung

Der historische Ursprung von Selektionsbildern geht auf die in Abschnitt 7.1 eingeführte folgende Aufteilung klassischer ABAP-Programme zurück:

- mit `SUBMIT` gestartete Reports
- über Dialogtransaktionen aufgerufene Dialogprogramme

Während die Dialogprogrammierung Kenntnisse in der Erstellung von und im Umgang mit Dynpros voraussetzte, sollte ein Report-Programmierer vollständig ohne die Verwendung des Screen Painter auskommen. Mit Selektionsbildern konnte auf sehr einfache Weise eine Benutzerschnittstelle realisiert werden, die den Ansprüchen des Reporting genügte. D.h., im Wesentlichen wurden Selektionskriterien für Datenbankselektionen abgefragt – daher auch der Name Selektionsbild.

Logische Datenbank

Logische Datenbanken, die im klassischen Reporting mit ausführbaren Programmen verknüpfbar waren, können sogar selbst Selektionsbilder als Komponenten enthalten. Bei der Verknüpfung einer solchen logischen Datenbank mit einem ausführbaren Programm übernimmt das Programm deren Selektionsbild. Dadurch entfiel dann sogar noch die Notwendigkeit der Definition eines eigenen Selektionsbildes im Programm.

Standard-selektionsbild

Bis Release 4.0 konnten Selektionsbilder tatsächlich nur in ausführbaren Programmen definiert werden. Bis dahin war auch nur ein einziges Selektionsbild (das Standardselektionsbild) pro Programm möglich, das beim Programmstart mit `SUBMIT` automatisch aufgerufen wurde (siehe Abbildung 7.1 in Kapitel 7).

Schon seit Release 4.0 können Selektionsbilder aber in allen Programmen definiert werden, die auch Dynpros enthalten können, also in Funktionsgruppen, ausführbaren Programmen und Modul-Pools, und jedes Programm kann mehrere Selektionsbilder enthalten. Die Selektionsbilder eines Programms sind nichts anderes als weitere Dynpros eines Programms. Sie haben eine eindeutige Dynpro-Nummer und können sehr ähnlich wie Dynpros aufgerufen werden.

Funktionsum

Die Bildschirmelemente von Selektionsbildern stellen eine Untermenge aller auf Dynpros möglichen Elemente dar und umfassen:

- Textfelder und Rahmen
- Ein-/Ausgabefelder, Ankreuzfelder und Auswahlknöpfe
- Drucktasten
- Subscreens
- Tabstrip Controls

Table Controls und Custom-Controls sind nicht möglich.[24] Die Unterstützung automatischer Eingabehilfen und -überprüfungen ist ähnlich wie bei allgemeinen Dynpros. Es können aber keine Dynpro-Folgen durch die Definition eines Folge-Dynpros gebildet werden.

24 Wenn unbedingt nötig, können aber allgemeine Subscreen-Dynpros eingebunden werden, die alle möglichen Bildschirmelemente enthalten.