

8 Ereignisse und Eventhandler

Wie bereits angedeutet, so verfügt bereits das einfache Picture-Control über Ereignisse, die es auslösen kann. Sie werden z.B. ausgelöst, wenn der Anwender im Anzeigebereich des Bildes einen Klick oder einen Doppelklick ausführt. Unser ABAP-Programm aus dem vorherigen Kapitel kann dann auf diese Ereignisse reagieren, indem es z.B. die eine oder andere Methode aufruft, um die Anzeigeeigenschaften des Bildes zu verändern.

Doch bevor wir diese Ereignissteuerung realisieren können, müssen noch einige grundlegende Erklärungen zur Art der Ereignisbehandlung im Control-Framework besprochen werden.

So werden die OO-Events des Controls nicht etwa direkt vom Control an unser ABAP-Programm durchgereicht, sondern erst einmal vom Control-Framework abgefangen, gepuffert und gefiltert, bevor der Eventhandler des eigentlichen Anwendungsprogramms zum Zuge kommt.

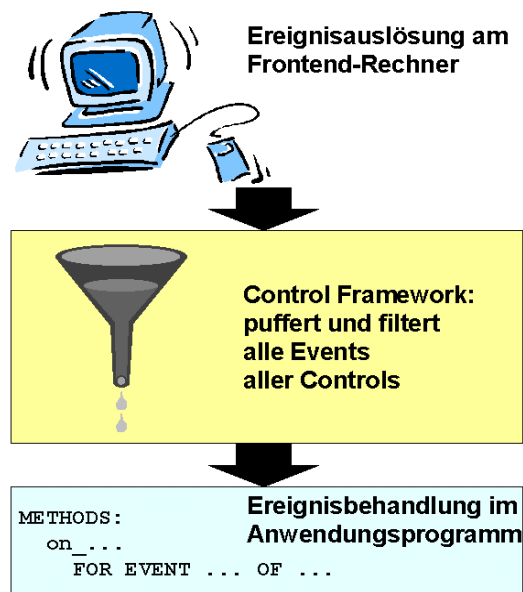


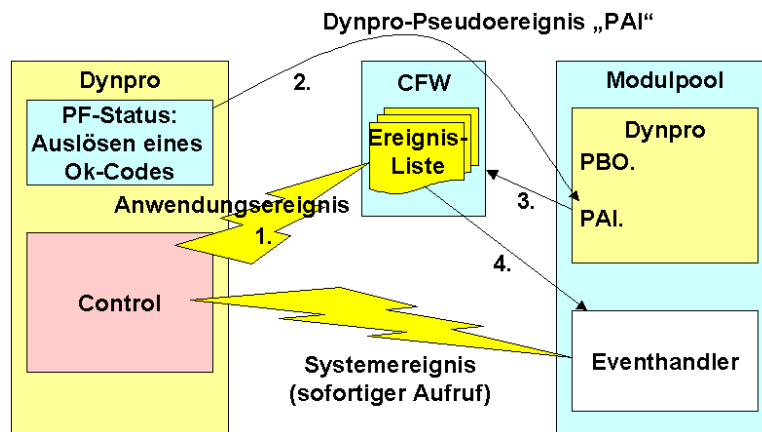
Abb. 8-1
Das Control-Framework
als Ereignispuffer

8.1 Systemereignisse und Anwendungereignisse

Controls können Ereignisse grundsätzlich auf zwei verschiedene Weisen an das Anwendungsprogramm übermitteln:

- als Systemereignis oder
- als Anwendungereignis.

Abb. 8-2
Systemereignisse und
Anwendungereignisse:
Reihenfolge der
Abarbeitung eines
Anwendungereignisses



8.1.1 Systemereignis

Ein Systemereignis ist die Form, wie man sich gewöhnlich ein Ereignis vorstellt, es trifft relativ unerwartet ein, wann immer es will. Dabei wird kein Dynpro-Ereignis (PAI) ausgelöst und somit auch keine Ablauflogik durchlaufen (PAI/PBO).

Dies hat weit reichende Folgen: Da PAI nicht durchlaufen wird, sondern nur der Eventhandler selbst, bekommt das Dynpro von dem Ganzen rein gar nichts mit!

Doch damit findet auch **kein** Feldtransport statt: Der Eventhandler hat keinerlei Zugriff auf ggf. vor Auslösen des Ereignisses getätigte Änderungen an Dynpro-Feldern!

Doch wie Sie SAP kennen, so gibt es auch an dieser Stelle einen Ausweg, falls Sie einen Bildwechsel erzwingen wollen.

Durch Aufruf der statischen Methode `set_new_okcode` der Klasse `cl_gui_cfw` können Sie aus dem Eventhandler des Systemereignisses heraus eine PAI-Verarbeitung anstoßen:

```
CALL METHOD cl_gui_cfw=>set_new_ok_code
EXPORTING
  new_code = okcode
IMPORTING
  rc = ...
```

Dieser Aufruf kann nur innerhalb eines Systemereignisses erfolgen!

In *rc* wird zurückgegeben, ob der Aufruf erfolgreich war:

- `cl_gui_cfw=>rc_posted`: Okcode erfolgreich gesetzt
- `cl_gui_cfw=>rc_invalid`: Ungültiger Okcode übergeben
- `cl_gui_cfw=>rc_wrong_state`: Es ist kein Systemereignis aktiv

8.1.2 Anwendungereignis

Ein als Anwendungereignis registriertes Ereignis wird nicht sofort, sondern erst mit der nächsten PAI-Verarbeitung an den Applikationsserver geschickt und dann verarbeitet. Dies bedeutet, dass zuvor die Dynproprüfungen durchlaufen werden und im Eventhandler die ggf. auf dem Dynpro geänderten Feldinhalte zur Verfügung stehen. Allerdings wird durch das Auslösen des Control-Ereignisses nicht gleich PAI ausgelöst, sondern lediglich das Ereignis für die nächste PAI-Verarbeitung »vorgemerkt«. Dies geschieht so:

Das Control-Framework (auf Seite des Applikationsservers) hält eine Tabelle mit den vom Control ausgelösten Ereignissen. Diese wurden vom Automation-Controller (auf Seite des Präsentationsservers) gefüllt. Dort »lungern die Ereignisse herum«, bis vom Applikationsserver deren Abarbeitung angestoßen wird.

Ein Anwendungereignis könnte man somit als »asynchrones Ereignis« bezeichnen. Denn mehrere verschiedene Anwendungereignisse können ausgelöst werden, bevor unser ABAP-Programm davon etwas mitbekommt.

Falls Sie Anwendungereignisse verwenden, so müssen Sie demnach zum Zeitpunkt PAI sicherstellen, dass Ihr Eventhandler zum Zuge kommt. Dies erreichen Sie durch Aufruf der statischen Methode *dispatch* der Klasse *cl_gui_cfw*, die Sie in einem Ihrer PAI-Module aufrufen.

```
CALL METHOD cl_gui_cfw=>dispatch
IMPORTING
    return_code = ...
```

Folgende Returncodes sind realisiert:

- `cl_gui_cfw=>rc_found`
Anwendungereignisse konnten an die Eventhandler abgesetzt werden.
- `cl_gui_cfw=>rc_unknown`
Das Ereignis war nicht registriert.

- `cl_gui_cfw=>rc_noevent`
Es gab kein Anwendungsereignis, es liegt ein »normaler« Okcode vor.
- `cl_gui_cfw=>rc_nodispatch`
Es war kein Eventhandler auf das Ereignis abonniert.

Wie Sie angeben können, welcher Ereignistyp ausgelöst werden soll, wird im entsprechenden Abschnitt bei der Registrierung des jeweiligen Ereignisses beschrieben.

8.1.3 Dynpro-Ereignisse – Control-Ereignisse

Im Zusammenspiel von Control- und Dynpro-Ereignissen tritt die unterschiedliche Philosophie der beiden Welten am stärksten zutage. Gerade in der Ereignisbehandlung ist es zwingend erforderlich, beide in ein harmonisches Zusammenspiel zu bringen. Wenn Sie sich eine komplexe Umgebung mit mehreren Controls vorstellen, wo einige Ereignisse als Systemereignisse, andere als Anwendungsereignisse ausgelöst werden, so können Sie sich vielleicht ausmalen, dass selbst SAP-Programmierer bei solchen Konstellationen einmal den Überblick verlieren und die eine oder andere Anwendung seltsame Ergebnisse bringt.

Doch es ginge auch anders – konsequent objektorientiert: Jeder Funktionscode aus dem PF-Status oder Dynpro ist ein Control (mit oder ohne Schaltfläche, mit oder ohne Menüeintrag, mit oder ohne Funktionstaste). Diese Controls besitzen ein einziges Ereignis, nämlich dass sie aktiviert wurden, und lösen ein Systemereignis aus. Dieses führt zum Aufruf des Okcode-Eventhandlers.

Einziges Problem bleiben die Datenfelder auf den Dynpro, sind auch sie Controls mit Attributen und Ereignissen? Müsste der Okcode-Eventhandler eine Methode eines solchen Feldcontrols aufrufen, um einen Feldtransport zu erreichen?

Dies wäre das Ende der klassischen SAP-Dialogprogrammierung. Wir wollen einmal sehen, was uns an dieser Stelle die Zukunft aus Walldorf bringen wird...

8.2 Registrierung und Abonnement

Wie Sie bereits aus den Grundlagen zu ABAP Objects wissen, reicht es nicht, dass einfach nur das Control ein Ereignis auslöst, sondern unser Anwendungsprogramm muss dieses Ereignis beim Control Framework *registrieren* und dann das Ereignis *abonnieren*.

8.2.1 Registrierung beim Control Framework

Nun können zahlreiche Controls wie auch unser Picture Control mehrere unterschiedliche Ereignisse auslösen. Dazu muss dem Control Framework mitgeteilt werden, welche Ereignisse es in Richtung des Anwendungsprogramms »abfeuern« soll. Dies geschieht sinnigerweise im Rahmen einer internen Tabelle, die per Methodenaufruf an das Control übergeben wird.

Die Event-Tabelle *events* besitzt folgenden Aufbau:

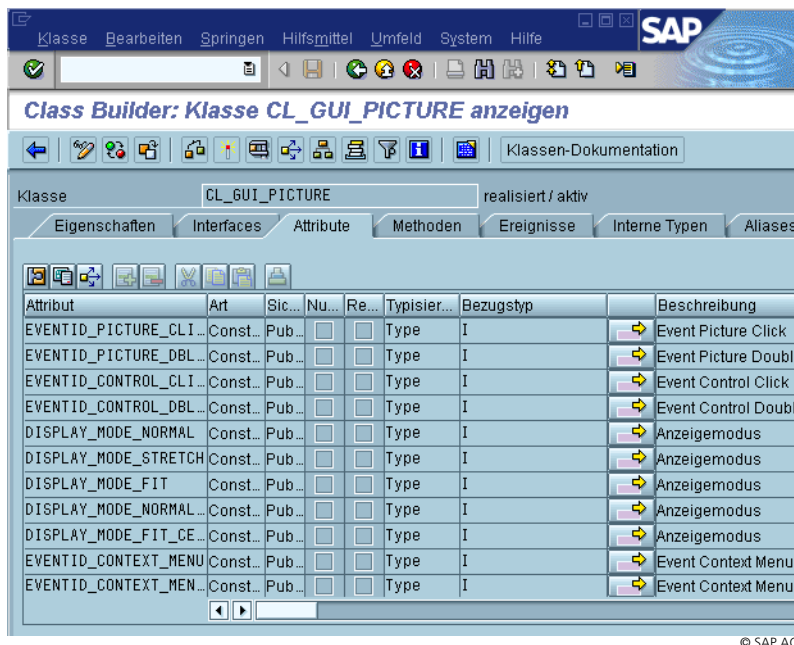
```
DATA events TYPE cntl_simple_events.
DATA event TYPE cntl_simple_event.
```

Wie in ABAP Objects vorgeschrieben, verfügt die interne Tabelle über keine Kopfzeile, daher haben wir eine entsprechende Workarea deklariert.

Der Datentyp *cntl_simple_event* besteht aus genau zwei Komponenten, der *event_id* des Ereignisses und einer Kennung *appl_event*, die angibt, ob das Ereignis als Systemereignis (Leerzeichen) oder als Applikationsereignis (>X<) ausgelöst werden soll.

Sie vermuten richtig: Die Ereignistabelle des CFW wird bereits bei der Registrierung aufgebaut, nicht erst bei Abonnement oder Auslösung des Ereignisses.

Die *Event-Ids* des Picture-Controls finden Sie als statische Konstanten-Attribute der Klasse *cl_gui_picture*.



Attribut	Art	Sic...	Nu...	Re...	Typisier...	Bezugstyp	Beschreibung
EVENTID_PICTURE_CLI...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Event Picture Click
EVENTID_PICTURE_DBL...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Event Picture Double
EVENTID_CONTROL_CLI...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Event Control Click
EVENTID_CONTROL_DBL...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Event Control Double
DISPLAY_MODE_NORMAL	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Anzeigemodus
DISPLAY_MODE_STRETCH	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Anzeigemodus
DISPLAY_MODE_FIT	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Anzeigemodus
DISPLAY_MODE_NORMAL...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Anzeigemodus
DISPLAY_MODE_FIT_CE...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Anzeigemodus
EVENTID_CONTEXT_MENU	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Event Context Menu
EVENTID_CONTEXT_MEN...	Const...	Pub...	<input type="checkbox"/>	<input type="checkbox"/>	Type	I	Event Context Menu

Abb. 8-3
Ereignisse des
Picture-Controls

Um z.B. den einfachen Klick auf das Bild als Systemereignis abzufangen, verwenden wir folgendes Coding:

```

REFRESH events.
event-eventid = picture->eventid_picture_click.
event-appl_event = ' '.
APPEND event TO events.
* ggf. weitere Event-Ids hinzufügen

CALL METHOD picture->set_registered_events
EXPORTING
    events = events.

```

8.2.2 Realisierung eines Eventhandlers

Um nun auch auf ein Ereignis reagieren zu können, muss ein Eventhandler realisiert werden und dieser muss auch das betreffende Ereignis abonnieren. In den meisten Fällen wird hierzu eine lokale Klasse erstellt, die dann statische Methoden als Eventhandler besitzt.

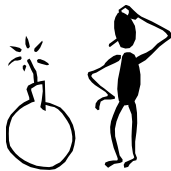
```

CLASS lcl_event_receiver DEFINITION.
    PUBLIC SECTION.
        CLASS-METHODS event_handler_picture_click
            FOR EVENT picture_click OF cl_gui_picture
            IMPORTING
                mouse_pos_x
                mouse_pos_y
                sender.
ENDCLASS.

CLASS lcl_event_receiver IMPLEMENTATION.
    METHOD event_handler_picture_click.
        IF sender = picture.
            * lassen Sie sich etwas einfallen...
        ENDIF.
    ENDMETHOD.
ENDCLASS.

```

Das Ereignis liefert uns neben der aktuellen Mausposition als *sender* die Instanzreferenz zum betreffenden Picture-Control (es könnten nämlich mehrere auf dem Dynpro vorhanden sein). Somit könnte eine unterschiedliche Ereignisbehandlung für jedes der Bilder realisiert werden.



8.2.3 Abonnieren eines Ereignisses

Nachdem der Eventhandler fertig ist, kann er das Ereignis abonnieren. Dies tun wir dort, wo wir das Picture-Control instanziiert haben (PBO).

```

SET HANDLER
    lcl_event_receiver=>event_handler_picture_click
    FOR picture.

```

8.2.4 Zusammenfassung

Um eine korrekte Ereignisbehandlung mit EnjoySAP Controls zu realisieren, müssen immer alle drei Schritte programmiert werden:

- **Registrierung** beim Control-Framework (als Anwendungs- oder Systemereignis)
- **Realisierung** des Eventhandlers (Anwendungslogik)
- **Abonnement** des Ereignisses (»Scharfmachen«)



Im obigen Beispiel wurde der Event-Handler als statische Methode einer lokalen Klasse realisiert. Dies muss nicht so sein. Alternativ kann auch eine Instanzmethode gewählt werden. Aus der Anwendungslogik ergeben sich meist jeweils Gründe für die eine oder andere Variante. Bei der statischen Methode erübrigt sich auf jeden Fall eine Instanziierung der Klasse.

8.3 Ein Kontextmenü auf dem Picture-Control

Seit SAP-Release 4.6 besteht die Möglichkeit, jedem beliebigen Dynpro-Element ein Kontextmenü zuzuordnen. Die Wirkung können Sie beobachten, indem Sie auf verschiedenen Dynpro-Feldern rechtsklicken. Leider ist diese Technik in den SAP-Dialogprogrammen noch sehr wenig verbreitet, besteht doch die Möglichkeit, jedem Element ein anderes, objektbezogenes Menü zu verpassen. Bei Auswahl eines solchen Menüeintrags wird ein entsprechender Okcode ausgelöst.

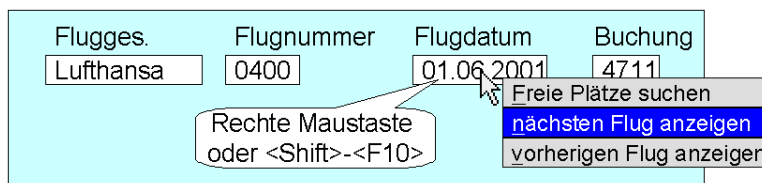


Abb. 8-4

Einstiegsbild zur Anlage einer Klasse

Da auch unser Picture-Control ein Dynpro-Element ist, kann auch dieses auf einen Klick mit der rechten Maustaste reagieren.

Ein Kontextmenü ist ein Oberflächenstatus vom Typ *Kontextmenü* und umfasst lediglich den Menübaum. Dort sind wie bei jedem PF-Status Funktionscodes hinterlegt.

Bei Drücken der **rechten** Maustaste über einem Bildelement mit zugeordnetem Kontextmenü oder Drücken der Taste <Shift>-<F10> erscheint dann das betreffende Kontextmenü, **ohne** dass dabei das Ereignis *PROCESS AFTER INPUT* ausgelöst wird.

8.3.1 Anlegen eines Kontextmenüs zur Laufzeit

Anders als bei der klassischen Dialogprogrammierung wird das Kontextmenü **nicht** mit dem Menu-Painter als PF-Status mit Typ *Kontextmenü* zum Designzeitpunkt erzeugt, sondern erst zur Laufzeit über das Ereignis »rechte Maustaste gedrückt« des Controls.

Dies ist sehr wichtig: Beim Drücken der rechten Maustaste wird ein Ereignis ausgelöst und der Eventhandler erzeugt dadurch erst das Menü! Die Auswahl eines Menüeintrages erfolgt viel später und hat mit der Menüerzeugung, die wir hier behandeln, überhaupt nichts zu tun! Wir haben es demnach bei einem Kontextmenü mit **zwei verschiedenen Ereignissen**, die nacheinander ausgelöst werden, zu tun; lesen Sie daher diesen Abschnitt sehr aufmerksam.

Um nun das Kontextmenü einmalig zu erzeugen, registrieren wir zunächst das Ereignis *eventid_context_menu* – wie bereits gelernt – beim CFW. Anschließend erweitern wir unsere lokale Klasse um einen entsprechenden Eventhandler. Einziger Importing-Parameter ist der *sender*.

In der Implementierung haben wir es mit einer neuen Klasse zu tun: das Kontextmenü *cl_ctmenu*. Nachdem die Funktionscodes mit samt erklärenden Texten in das Kontextmenü eingetragen wurden, muss nun dem Control der Befehl gegeben werden, das Menü anzuzeigen (wir sind hier im Ereignis »rechte Maustaste gedrückt«).

```
METHOD event_handler_context_menu.
  DATA: menu TYPE REF TO cl_ctmenu.

  CREATE OBJECT menu.
  CALL METHOD menu->add_function
    EXPORTING
      fcode = 'FIT'
      text = 'Bild vergrößern'.
  CALL METHOD menu->add_function
    EXPORTING
      fcode = 'NORMAL'
      text = 'Originalgröße'.
  CALL METHOD sender->display_context_menu
    EXPORTING
      context_menu = menu.
ENDMETHOD.
```

Der SAP-Style-Guide sieht vor, dass Sie in ein Kontextmenü nur solche Funktionscodes aufnehmen, die auch im »normalen« PF-Status des aktiven Dynpros vorkommen. Auch soll ein Kontextmenü möglichst nicht mehr als zehn Einträge besitzen und nicht allzu tief geschachtelt sein.

Selbstverständlich muss der Eventhandler noch das entsprechende Ereignis abonnieren:

```
SET HANDLER
  event_receiver->event_handler_context_menu
  FOR picture.
```

8.3.2 Reaktion auf die Auswahl einer Funktion

Auch bei Auswahl einer Funktion aus dem Kontextmenü wird ein Ereignis ausgelöst. Somit muss auch das Ereignis *context_menu_selected* des Picture-Controls registriert, der Eventhandler realisiert und das Ereignis abonniert werden.

Das Ereignis liefert uns außer dem *sender* noch den ausgewählten Funktionscode als Parameter *fcode*.

Die Implementierung könnte dann z. B. so aussehen:

```
METHOD event_handler_context_menu_sel.
  DATA: display_mode TYPE I.

  CASE fcode.
    WHEN 'FIT'.
      display_mode =
        CL_GUI_PICTURE=>display_mode_fit_center
    WHEN 'NORMAL'.
      display_mode =
        CL_GUI_PICTURE=>display_mode_normal_center
  ENDCASE.
  CALL METHOD sender->SET_DISPLAY_MODE
  EXPORTING
    DISPLAY_MODE = display_mode.
ENDMETHOD.
```

→ Auch hier darf das Abonnement des Ereignisses nicht vergessen werden!

8.4 Zusammenfassung

Die hier anhand des Picture Controls und der Kontextmenüs gezeigten Mechanismen der Ereignisregistrierung, -behandlung und -abonnement werden genauso ebenfalls bei allen anderen Controls benutzt. In den folgenden Kapiteln wird lediglich auf die Möglichkeiten eingegangen, ohne deren Anwendung im Detail zu beschreiben.

Das SAP-Ereignismanagement des Control Frameworks ist somit eine der wichtigsten Errungenschaften der Objektorientierung in ABAP.



8.5 Übungsaufgaben

Aufgabe 1

Programmname: ZAO00081

Kopieren Sie das Übungsprogramm von Kapitel 7. Beim Klick auf das Bild soll dieses maximiert werden (*display_mode_fit*).

Aufgabe 2

Programmname: ZAO00082

Erweitern Sie das Bildanzeigeprogramm: Der Anzeigemodus soll nun auch über ein Kontextmenü auswählbar sein (*display_mode_normal/ display_mode_fit*).