

# Controls Tutorial (BC-CI)



**Release 4.6C**



## Copyright

© Copyright 2001 SAP AG. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Software-Produkte können Software-Komponenten auch anderer Software-Hersteller enthalten.

Microsoft<sup>®</sup>, WINDOWS<sup>®</sup>, NT<sup>®</sup>, EXCEL<sup>®</sup>, Word<sup>®</sup>, PowerPoint<sup>®</sup> und SQL Server<sup>®</sup> sind eingetragene Marken der Microsoft Corporation.

IBM<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, DB2/6000<sup>®</sup>, Parallel Sysplex<sup>®</sup>, MVS/ESA<sup>®</sup>, RS/6000<sup>®</sup>, AIX<sup>®</sup>, S/390<sup>®</sup>, AS/400<sup>®</sup>, OS/390<sup>®</sup> und OS/400<sup>®</sup> sind eingetragene Marken der IBM Corporation.

ORACLE<sup>®</sup> ist eine eingetragene Marke der ORACLE Corporation.

INFORMIX<sup>®</sup>-OnLine for SAP und Informix<sup>®</sup> Dynamic Server<sup>™</sup> sind eingetragene Marken der Informix Software Incorporated.

UNIX<sup>®</sup>, X/Open<sup>®</sup>, OSF/1<sup>®</sup> und Motif<sup>®</sup> sind eingetragene Marken der Open Group.

HTML, DHTML, XML, XHTML sind Marken oder eingetragene Marken des W3C<sup>®</sup>, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc.

JAVASCRIPT<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo und mySAP.com sind Marken oder eingetragene Marken der SAP AG in Deutschland und vielen anderen Ländern weltweit. Alle anderen Produkte sind Marken oder eingetragene Marken der jeweiligen Firmen.

## Symbole

Symbol	Bedeutung
	Achtung
	Beispiel
	Empfehlung
	Hinweis
	Syntax
	Tip

## Inhalt

<b>Controls Tutorial (BC-CI)</b> .....	<b>5</b>
BC - Component Integration: Controls Tutorial .....	6
Benutzerhinweise .....	7
Voraussetzungen.....	8
Systemkonfiguration .....	9
Architektur.....	10
<b>Lektion 1: Anlegen eines Custom Controls</b> .....	<b>12</b>
Einführung.....	13
Übung 1: Einen Bereich für ein Control reservieren.....	15
Übung 2: Ein Control und dessen Container erzeugen.....	17
Übung 3: Methoden des Controls aufrufen.....	20
Zusammenfassung .....	22
<b>Lektion 2: Ereignisbehandlung</b> .....	<b>23</b>
Einführung.....	26
Übung 1: Eine Ereignisbehandlungsmethode schreiben .....	28
Übung 2: Ein Anwendungs-Ereignis registrieren .....	30
Übung 3: Ein System-Ereignis registrieren .....	32
Übung 4: Auf ein Ereignis reagieren .....	34
Zusammenfassung .....	37
<b>Lektion 3: Flushoptimierung</b> .....	<b>38</b>
Einführung.....	39
Übung 1: Importierte Werte verwenden .....	42
Übung 2: Wertübergabe zwischen Methoden.....	45
Übung 3: Control Methoden in Unterprogrammen .....	48
Übung 4: Mehrere Controls verwenden .....	50
Zusammenfassung .....	52
<b>Lektion 4: Testen und Fehlerbehandlung</b> .....	<b>53</b>
Einführung.....	54
Übung 1: Den Trace Modus aktivieren .....	56
Übung 2: Synchroner Verarbeitung der Methoden .....	58
Zusammenfassung .....	59
Weitere Informationsquellen .....	60

## Controls Tutorial (BC-CI)

### Was sind Controls?

Controls – im Deutschen wird auch der Begriff *Steuerelemente* verwendet – sind eigenständige binäre Software-Komponenten, die wiederverwendbar sind. Entwickler können ein oder mehrere Controls in ihre Anwendungen integrieren und deren bereitgestellte Funktionen nutzen. Ein typisches Beispiel ist ihr Einsatz bei der Gestaltung der Benutzeroberfläche. Die Anwendung von Controls beschränkt sich aber nicht nur auf sichtbare Komponenten.

### Wer hat mit Controls zu tun?

#### Der Anwender...

...sieht nicht die verschiedenen Komponenten, aus denen eine Anwendung zusammengesetzt ist. Daher nimmt er die Integration von Controls auch nicht wahr. Für ihn bleibt es eine einzelne Anwendung, mit der er arbeitet.

#### Der Anwendungsentwickler...

...benutzt Controls in seinen Anwendungen. Ihn interessieren keine technischen Details. Bei der Programmierung greift er auf eine logische Verschaltung zu, die ihm bei der Integration des Controls hilft.

#### Der Verschaler...

...programmiert die logische Verschaltung von einzelnen Controls. Dazu muß er tiefgehendes technisches Wissen über Controls und des SAP eigenen *Control Framework* mitbringen.

#### Der Control-Entwickler...

...entwickelt die Controls selbst. Die angestrebte Wiederverwendbarkeit setzt eine standardisierte Schnittstelle voraus, nach der sich der Control-Entwickler richten muß.

### Wem nutzt dieses Tutorial?

Diese Dokumentation richtet sich an den *Anwendungsentwickler*, die Controls in ihre ABAP Programme einbauen wollen.

---

**BC - Component Integration: Controls Tutorial**

## BC - Component Integration: Controls Tutorial

Die Dokumentation *BC - Component Integration: Controls Tutorial* ermöglicht einen Einstieg für Entwickler, die verschaltete Controls in ABAP-Programmen nutzen wollen. Das Tutorial beschränkt sich dabei auf sogenannte Custom Controls, mit denen man moderne Oberflächenelemente in Anwendungen nutzen kann.

Das Kapitel [Benutzerhinweise \[Seite 7\]](#) klärt neben den technischen Voraussetzungen, welche Programmierkenntnisse nötig sind, um das Tutorial durcharbeiten zu können. Es gibt zudem eine kurze Übersicht, wie Controls in das R/3-System integriert sind.

Die Übungen des Tutorials bauen aufeinander auf. Sie sind in Lektionen eingeteilt, die jeweils ein abgestecktes Thema behandeln:

[Lektion 1: Anlegen eines Custom Controls \[Seite 12\]](#)

[Lektion 2: Ereignisbehandlung \[Seite 23\]](#)

[Lektion 3: Flushoptimierung \[Seite 38\]](#)

[Lektion 4: Testen und Fehlerbehandlung \[Seite 53\]](#)

Zu jedem abgedeckten Thema gibt es eine Einführung mit Hintergrundinformationen.

## Benutzerhinweise

Bitte lesen Sie die folgenden Hinweise, bevor Sie das Tutorial durcharbeiten:

[Voraussetzungen \[Seite 8\]](#)

[Systemkonfiguration \[Seite 9\]](#)

[Architektur \[Seite 10\]](#)

## Voraussetzungen

# Voraussetzungen

Das Tutorial macht Sie Schritt für Schritt mit Konzepten der Control-Programmierung vertraut. Praktische Kenntnisse in der ABAP Programmierung und im Umgang mit den Werkzeugen der ABAP Workbench werden vorausgesetzt. Informationen hierzu entnehmen Sie bitte der Dokumentation:

[BC - ABAP Workbench Tutorial \[Extern\]](#)

[BC - ABAP Workbench: Werkzeuge \[Extern\]](#)

Auf Controls wird über Klassen von *ABAP Objects* – die objektorientierte Erweiterung von ABAP - zugegriffen. Eine Einführung in die objektorientierte Programmierung hätte allerdings den Rahmen dieses Tutorials gesprengt. Dennoch vermitteln die **Grundlagen**-Abschnitte ausgewählter Lektionen ABAP Objects Konzepte, soweit sie für das Verständnis der Controls-Programmierung nötig sind. Sie können diese Abschnitte bei entsprechenden Vorkenntnissen überspringen.



Eine Einführung zum Thema Objektorientierung in ABAP finden Sie in [BC - ABAP Programmierung \[Extern\]](#) im Kapitel [ABAP Objects \[Extern\]](#).

## Systemkonfiguration

Die Programmierung von Controls wird ab Release 4.0A über Funktionsbausteine unterstützt. Seit Release 4.5A ist mit *ABAP Objects* die objektorientierte Programmierung mit ABAP möglich. Eine objektorientierte Version des Control Frameworks ist mit Release 4.6A verfügbar. Die Programmbeispiele in diesem Tutorial setzen dieses Release voraus.

Zusätzlich spielen zwei weitere Faktoren eine Rolle:

- Das verwendete Betriebssystem auf dem Frontend PC.
- Die installierte Version des Controls auf dem Frontend PC.

### Frontend Plattform

Controls werden über eine standardisierte Schnittstelle angesprochen. Auf dem Markt werden verschiedene solcher Standards angeboten, von denen zwei vom System unterstützt werden: *ActiveX* und *JavaBeans*.

Bis Release 4.5B müssen Sie über Funktionsbausteine in Ihrem ABAP-Programm prüfen, ob das Frontend den entsprechenden Standard unterstützt. Ab Release 4.6A übernimmt das *Control Framework* diese Überprüfung automatisch und löst im Fehlerfall eine Exception aus. Benutzen Sie die Funktionsbausteine, wenn Sie individuell auf diesen Fehler reagieren möchten - zum Beispiel weil noch keine JavaBean-Implementierung des Controls existiert.

#### Unterstützte Schnittstellen-Standards

	Windows95 (32 Bit) WindowsNT	Andere Plattformen
Verwendeter Standard	ActiveX	JavaBean
Funktionsbaustein, der prüft, ob der Standard am Frontend verfügbar ist (nur bis Release 4.5B erforderlich)	HAS_GUI_ACTIVEX	HAS_GUI_JAVABEAN

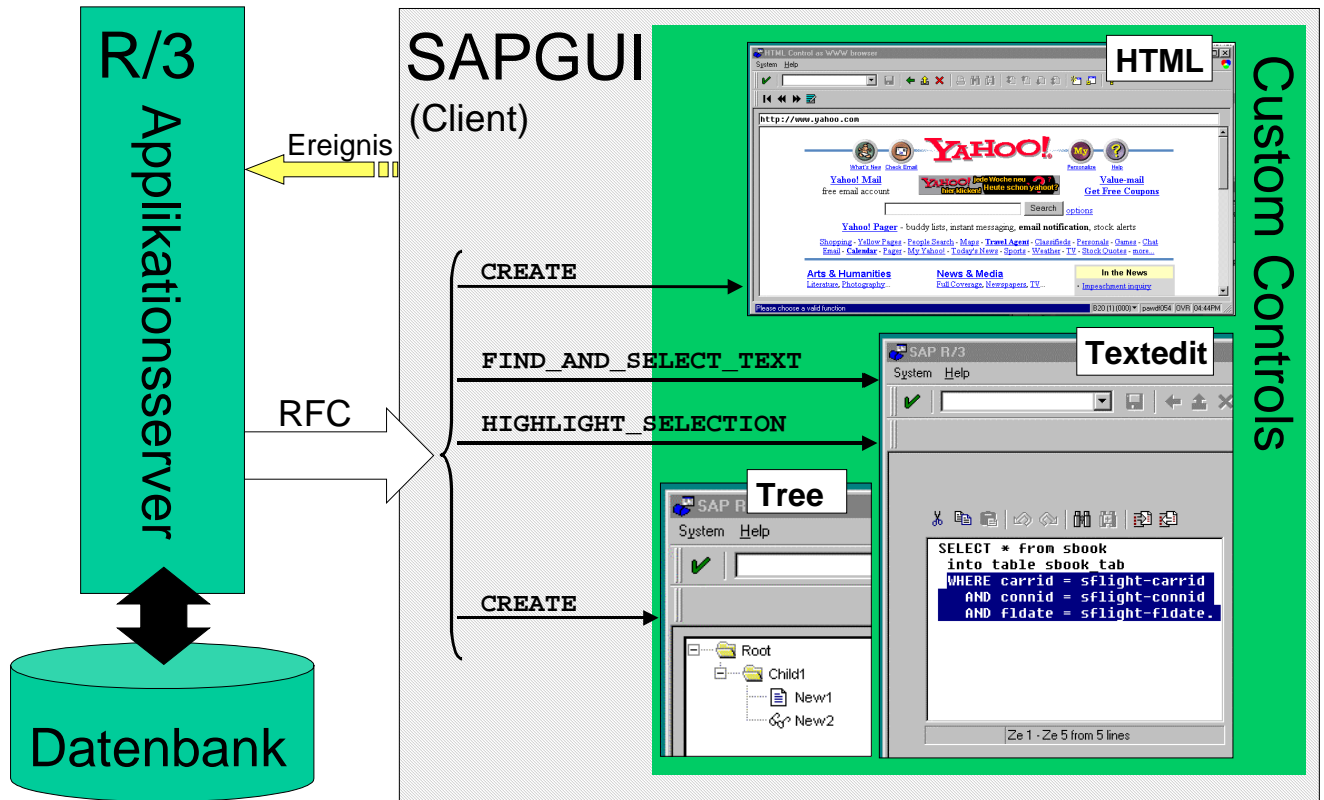
### Controls Version

Zusammen mit der lokalen SAPGUI-Installation werden auf dem Frontend PC die passenden Controls installiert. Wenn Sie einen neuen lokalen SAPGUI installieren, sollten Sie zuerst die alte Version deinstallieren, um einen konsistenten Zustand zu garantieren.

Architektur

Architektur

R/3-Architektur



Erläuterung

Bei der Programmierung von Controls ist hauptsächlich das Zusammenspiel von Applikationsserver (Backend) und SAPGUI (Frontend) von Bedeutung. Sie können auf dem Frontend installierte Controls über den Aufruf von ABAP-Methoden erzeugen, Control-Eigenschaften verändern und das Verhalten dieser Controls steuern. Diese Methoden werden vom Applikationsserver mit Hilfe eines Remote Function Call (RFC) zum Frontend übertragen und ausgeführt.

Der Benutzer eines Anwendungsprogramms löst durch bestimmte Aktionen (zum Beispiel durch einen Doppelklick auf einen Knoten in einer Baumstruktur) Control-Ereignisse aus. Sie können entscheiden, welche Ereignisse Sie im ABAP-Programm abfangen möchten und diese beim entsprechenden Control registrieren. Ein ausgelöstes Ereignis wird vom SAPGUI zum Applikationsserver übertragen. Sie können dann – als Reaktion auf das Ereignis – das weitere Verhalten des Controls mit weiteren Methodenaufrufen steuern.



---

**Lektion 1: Anlegen eines Custom Controls**

## Lektion 1: Anlegen eines Custom Controls

### Aufgabenstellung

In dieser Lektion schaffen Sie zunächst die Voraussetzungen, um ein Control in ein ABAP-Programm zu integrieren. Danach erzeugen Sie ein Control für die Bearbeitung von Texten, dessen Funktionalität Sie später Schritt für Schritt weiter ausbauen werden.

### Überblick : Lerninhalte

Sie lernen in dieser Lektion:

- Wie Sie ein sichtbares Control auf einem Dynpro integrieren.
- Wie Sie ein Control erzeugen.
- Welche Funktionen das Textedit Control automatisch unterstützt.

### Grundlagen

Controls sind über Klassen von ABAP Objects verschalt. Das Anlegen eines Control entspricht daher der Instanziierung eines Objekts, d.h. der Erzeugung einer Instanz.

Objekte werden generell über einen *Konstruktor* erzeugt, der erste Eigenschaften des Objekts bestimmt. Jede Klasse, die es erlaubt, Objekte zu erzeugen, hat einen solchen Konstruktor.

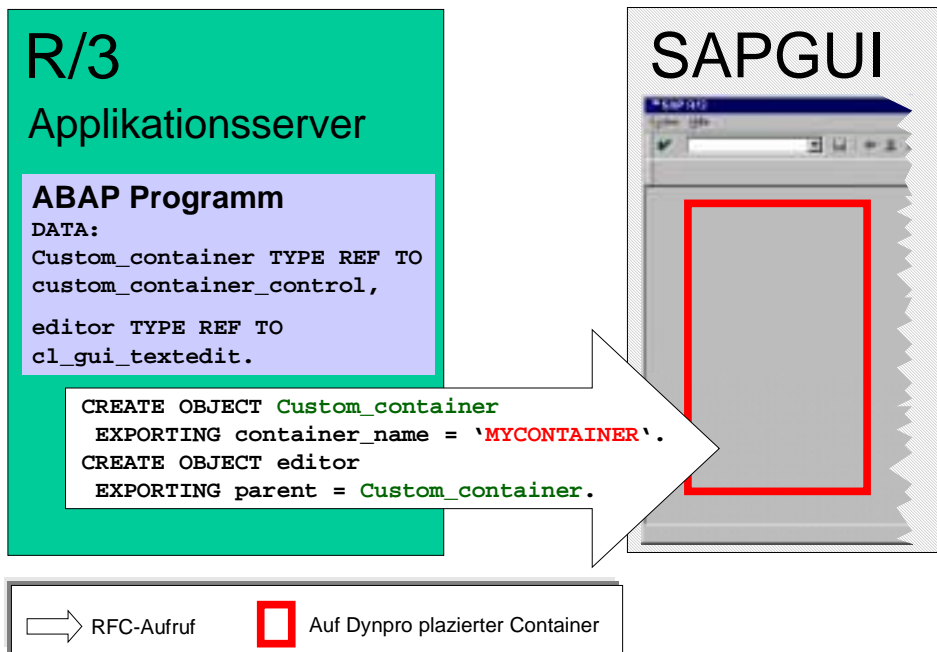
Bei der Erzeugung eines Objekts rufen Sie den Konstruktor nun nicht explizit auf. Statt dessen legen Sie eine Instanz mit der Anweisung `CREATE OBJECT` an, die den Konstruktor dann implizit ausführt.

### Fortsetzung

[Lektion 2: Ereignisbehandlung \[Seite 23\]](#) beschreibt, wie Sie Ereignisse eines Controls abfangen und weiterverarbeiten.

## Einführung

### R/3-Architektur



### Erläuterung

Jedes Control, das Sie auf einem Dynpro anzeigen wollen, müssen Sie einem *Container Control* zuordnen. In der oben angezeigten Grafik ist die Integration eines Textedit Controls mit Hilfe eines *Custom Container Control* dargestellt. Über einen im Screen Painter angelegten Container muß ein Bereich festgelegt werden, den das Custom Container Control ausfüllen soll.

Ein Container Control organisiert die Darstellung von ein oder mehreren Controls. Je nach Art des Container Controls können Sie:

- Ein Control fest auf dem Dynpro platzieren, zum Beispiel in einem Subscreen (*Custom Container Control*)
- Mehrere Controls in einen in Bereiche eingeteilten Container Control einbetten (*Splitter Container Control* und *Easy Splitter Container Control*)
- Ein Control in einem amodalen frei beweglichen Fenster darstellen (*Dialogbox Container Control*)
- Ein Control an einen ausgewählten Rand eines Dynpros 'andocken' (*Docking Container Control*).

## Einführung



Die einzelnen Container Controls sind in der Dokumentation über [SAP Container \[Extern\]](#) näher erläutert.

Zusätzlich lassen sich Container Controls schachteln, d.h. für den Parameter **parent** eines Container Controls kann wiederum ein Container Control angegeben werden. Auf diese Weise sind vielfältige Anordnungen von Controls auf dem Bildschirm möglich.

Sie können ein Container Control nach dessen Erzeugung auch auf einem anderen Dynpro anzeigen. Dazu müssen Sie dieses Control 'umhängen', d.h. mit einem anderen Dynpro verknüpfen. Controls, die dieses Container Control als **parent** angegeben haben, werden dann mitverschoben. Bei geschachtelten Container Controls können Sie nur das Control, in dem alle anderen enthalten sind, umhängen.



Jedes Dynpro ist einem Popup-Level von 0 bis 9 zugeordnet. Container Controls können während ihrer Lebensdauer keinem anderen Level zugeordnet werden als dem bei der Erzeugung angegebenen. Falls Sie ein Control auf einem anderen Popup-Level anzeigen wollen, müssen Sie also ein neues Container Control für diesen Level erzeugen.

## Folgerungen

Aus der Architektur ergeben sich folgende Arbeitsschritte zum Anlegen eines Control:

1. Für die Darstellung eines Control in einem spezifischen Bereich auf einem Dynpro legen Sie im *Screen Painter* einen Container an.
2. Mit der Anweisung **CREATE OBJECT** legen Sie zum PBO-Zeitpunkt ein Container Control an, das sie dem Dynpro zuordnen.
3. Mit der Anweisung **CREATE OBJECT** legen Sie zum PBO-Zeitpunkt eine Control-Instanz an, die sie dem Container Control zuordnen.

## Übung 1: Einen Bereich für ein Control reservieren

## Übung 1: Einen Bereich für ein Control reservieren

### Voraussetzungen

Sie haben bereits ein ausführbares Programm als lokales Objekt angelegt.

### Vorgehensweise

1. Ergänzen Sie den Quelltext Ihres Rahmenprogramms um die Zeilen:

```
START-OF-SELECTION  
SET SCREEN '100'.
```

und legen das Dynpro mit den folgenden Attributen an:

- *Dynprotyp*: Normal
- *Folgedynpro*: 100

2. Bearbeiten Sie nun das *Layout* Ihres angelegten Dynpro im *Screen Painter* und markieren Sie einen Bereich für Ihr Custom Control.

Wenn Sie den alphanumerischen Screen Painter verwenden, gehen Sie wie folgt vor:

- a. Plazieren Sie Ihren Cursor in eine Zeile des Dynpro und wählen Sie den Menüpunkt *Bearbeiten* → *Anlegen Element* → *Custom Control*.
- b. Um die untere rechte Ecke des Bereichs zu markieren, können sie entweder auf die entsprechende Zeile und Spalte doppelklicken oder die Drucktaste *Ctrl Ende mark.* benutzen. Es erscheint ein Fenster für die Elementattribute.
- c. Vergeben Sie den Namen **MYCONTAINER1** und *übernehmen* Sie Ihre Einstellungen.

Wenn Sie den graphischen Screen Painter verwenden, gehen Sie wie folgt vor:

- a. Klicken Sie auf das Custom Control Icon (zu erkennen an dem Buchstaben 'C') und markieren Sie einen Bereich auf dem Dynpro.
- b. Tragen Sie den Namen **MYCONTAINER1** in das Feld *Name* ein. Sie können den Namen auch im Fenster für die Attribute vergeben, das durch einen Doppelklick auf das angelegte Element erscheint.

3. Um das Bild nach Programmstart wieder verlassen zu können, erweitern Sie Ihr Dynpro um eine Drucktaste und fragen den Funktionscode im PAI Modul ab:

- a. Definieren Sie im *Layout Editor* des *Screen Painter* eine Drucktaste mit dem Funktionscode **EXIT** und plazieren Sie sie auf dem Dynpro.
- b. Tragen Sie in der Elementliste den Namen **OK\_CODE** für das Element vom Typ **OK** ein.
- c. Legen Sie ein PAI Modul **user\_command\_0100** an und fragen Sie dort den Funktionscode der Drucktaste ab:

```
MODULE USER_COMMAND_0100 INPUT.  
  CASE OK_CODE.  
    WHEN 'EXIT'.
```

**Übung 1: Einen Bereich für ein Control reservieren**

```
        LEAVE TO SCREEN 0.  
    ENDCASE.  
ENDMODULE.          " USER_COMMAND_0100 INPUT
```

- d. Deklarieren Sie `OK_CODE` als globales Datenfeld, indem Sie darauf doppelklicken und die folgende Zeile in das Rahmenprogramm übernehmen:

```
DATA ok_code LIKE sy-ucomm.
```

4. Speichern Sie Ihre Einstellungen und aktivieren Sie alle Ihre Objekte.

**Überprüfen Sie Ihre Arbeit**

Wenn Sie Ihr Programm starten erscheint Ihr Dynpro mit einer Drucktaste. Der Container für das Custom Control wird nicht angezeigt. Wenn Sie auf die Drucktaste klicken, kehren Sie wieder zu Ihrem Rahmenprogramm zurück.

## Übung 2: Ein Control und dessen Container erzeugen

# Übung 2: Ein Control und dessen Container erzeugen

## Voraussetzungen

Sie haben in Übung 1 für Ihr Rahmenprogramm ein Dynpro angelegt, auf dem Sie einen Bereich für ein *Custom Container Control* plziert haben. Nun erzeugen Sie dieses Control und verbinden es mit einem [TextEdit Control \[Extern\] \[Extern\]](#).

## Vorgehensweise

- Deklariieren Sie folgende Variablen in Ihrem Rahmenprogramm:
  - Eine Referenzvariable auf die Klasse des Custom Container Controls,
  - eine Referenzvariable auf die Klasse des Textedit Controls und
  - eine Variable, die die ID des aktuellen Reports enthält:

```
DATA: custom_container TYPE REF TO cl_gui_custom_container,
      editor TYPE REF TO cl_gui_textedit,
      repid LIKE sy-repid.
```
- Legen Sie eine Konstante `line_length` an, die die Länge einer Zeile im Textfenster des Textedit Controls festlegt:
 

```
CONSTANTS: line_length TYPE I VALUE 256.
```
- Legen Sie ein PBO Modul `status_100` in der Ablauflogik zum Dynpro 100 an.
- Im PBO Modul instanziiieren Sie das Custom Container Control, das Sie mit dem Custom Control Bereich `MYCONTAINER1` verknüpfen:
 

```
IF EDITOR IS INITIAL.
  repid = sy-repid.
  CREATE OBJECT custom_container
  EXPORTING
    CONTAINER_NAME = 'MYCONTAINER1'
  EXCEPTIONS
    CNTL_ERROR = 1
    CNTL_SYSTEM_ERROR = 2
    CREATE_ERROR = 3
    LIFETIME_ERROR = 4
    LIFETIME_DYNPRO_DYNPRO_LINK = 5.
```
- Danach instanziiieren Sie im PBO das TextEdit Control und legen für den Parameter `parent` das eben angelegte Container Control fest:
 

```
CREATE OBJECT editor
  EXPORTING
    parent = custom_container
    WORDWRAP_MODE = CL_GUI_TEXTEDIT=>WORDWRAP_AT_FIXED_POSITION
    WORDWRAP_POSITION = line_length
    WORDWRAP_TO_LINEBREAK_MODE = CL_GUI_TEXTEDIT=>TRUE.
ENDIF.      "editor is initial
```
- Aktivieren Sie das Rahmenprogramm.

## Übung 2: Ein Control und dessen Container erzeugen

### Überprüfen Sie Ihre Arbeit

Starten Sie Ihr Programm. Auf dem Bild erscheint nun an der Stelle, an der Sie den Container angelegt haben, ein Texteditor. Über dem Eingabefeld des Editors sind neun Icons sichtbar:



Sollten sie weniger Icons auf Ihrem Bild sehen, müssen Sie den Container im Screen Painter vergrößern.

Das Textedit Control stellt bereits grundlegende Funktionen für das Bearbeiten von Texten zur Verfügung:

- Ausschneiden, Kopieren und Einsetzen von Text
- Zurücknehmen von ausgeführten Aktionen
- Suchen und Ersetzen von Text
- Laden und Speichern von lokalen Dateien

Sie können diese Funktionen über die Iconleiste oder das Kontextmenü (rechte Maustaste) ausprobieren. Weiterhin stehen Ihnen Drag and Drop Funktionen zur Verfügung:

- Laden von Texten durch das Ziehen von Textfile-Icons (keinen Shortcut) in das Textfenster.
- Verschieben von im Textfenster markierten Text mit der rechten Maustaste.

## Diskussion

### Attribute

Die Anweisung **CREATE OBJECT** ruft den *Konstruktor* der Klasse `cl_gui_textedit` auf. Ein Konstruktor bestimmt erste Eigenschaften des instanziierten Objekts bei dessen Erzeugung. In der Übung werden zwei Attribute mit Hilfe von Konstanten der Klasse gesetzt. Das Attribut **WORDWRAP\_MODE** bestimmt, ob und wie ein Zeilenumbruch durchgeführt wird. Das Attribut **WORDWRAP\_TO\_LINEBREAK\_MODE** bestimmt, ob das System den automatisch durchgeführten Zeilenumbruch in einen harten Zeilenumbruch umwandelt, wenn der Text im R/3 System abgespeichert wird.



Methoden (inklusive Konstruktor), Attribute und Ereignisse einer Klasse können Sie sich im [Class Builder \[Extern\]](#) anzeigen lassen (über die Transaktion `se24` oder über den Menüeintrag *Entwicklung* → *Class Builder* vom Einstiegsbild der ABAP Workbench).

### Zeitpunkt der Erzeugung

Alle Methoden, die ein Control ansteuern, müssen per RFC zum Frontend übertragen werden. Sie können den Zeitpunkt, wann dies geschehen soll, explizit mit dem Aufruf der Methode `CL_GUI_CFW=>FLUSH` festlegen.

## Übung 2: Ein Control und dessen Container erzeugen



Am Ende von jedem PBO Modul wird diese Methode nach dem Feldtransport **implizit** aufgerufen. Dies garantiert, daß vor einem Bildaufbau die Methoden zum Frontend übertragen werden (siehe auch [Synchronisation der Automation Queue \[Extern\]](#) der Control Framework Dokumentation).

Die Aktualparameter von Control-Methoden müssen zum Zeitpunkt der Übertragung gültig sein. Die Werte von globalen Systemvariablen ändern sich häufig, da sie auch in aufgerufenen Methoden und Funktionsbausteinen verwendet werden. Folglich sind sie als Aktualparameter ungeeignet.

Aus diesem Grund wurde statt der Systemvariablen **sy-repid** die Variable **repid** in der Konstruktormethode übergeben, nachdem ihr der aktuelle Wert des Systemfeldes zugewiesen wurde. Die Variable **repid** ist im nur im Rahmenprogramm sichtbar und bleibt daher stabil.

### Lebensdauer

Alle Anweisungen sind in einem **IF** Block geklammert, damit das Control nur einmal instanziiert wird und nicht jedesmal beim Aufruf des PBO Moduls.

Die Lebensdauer eines Controls hängt vom Modus ab, den Sie mit dem Parameter **LIFETIME** im Konstruktor angegeben haben. In der Voreinstellung (Modus *IMode*) existiert ein einmal angelegtes Control bis:

- Der interne Modus, in dem es erzeugt wurde, abgebaut wird.
- Das Container Control, in dem es eingebettet ist, abgebaut wird.
- Sie die Methode **FREE** der angelegten Instanz aufrufen.



Genauerer zur Lebensdauer von Controls finden Sie im Abschnitt [Lifetime Management \[Extern\]](#) der Control Framework Dokumentation.

## Übung 3: Methoden des Controls aufrufen

# Übung 3: Methoden des Controls aufrufen

## Verwendung

Das Verhalten einer einmal angelegten Instanz eines Control wird über Methoden gesteuert. Mit Hilfe dieser Methoden nutzen Sie Funktionen des Control und bestimmen dessen Eigenschaften. Um Control-Methoden auszuführen, müssen Sie sie in der Regel zum Frontend übertragen.

In dieser Übung implementieren Sie eine Funktion, die einen Text aus einer internen Tabelle in das Textfenster lädt.

## Vorgehensweise

- Um einen Text zu generieren, deklarieren Sie eine interne Tabelle und einen Arbeitsbereich global im Rahmenprogramm. Deklarieren Sie außerdem ein globales Flag `g_loaded`:

```
TYPES: begin of mytable_line,
       line(line_length) type C,
       end of mytable_line.
DATA: MYTABLE TYPE TABLE OF MYTABLE_LINE,
      TEXTSTRUCT TYPE MYTABLE_LINE,
      g_loaded TYPE C.
```



Die Zeilenlänge der internen Tabelle wurde bewußt mit der Position des Zeilenumbruchs im Textedit-Control gleichgesetzt (siehe Methode [Übung 2: Ein Control und dessen Container erzeugen \[Seite 17\]](#)).

- Definieren Sie ein Textelement und füllen Sie Ihre interne Tabelle im PBO mit Text. Dies soll nur einmal geschehen und wird daher in den `IF`-Block zur Erzeugung des Controls mit aufgenommen:

```
DO 20 TIMES.
  WRITE TEXT-001 TO TEXTSTRUCT-LINE.
  APPEND TEXTSTRUCT TO MYTABLE.
ENDDO.
```

- Um die neue Funktion zugänglich zu machen, legen Sie auf dem Dynpro 100 eine weitere Drucktaste mit folgenden Eigenschaften an:
  - Funktionscode: `IMP`
  - Text: `Import`
  - Icon: `ICON_IMPORT`
- Fragen Sie im PAI Modul den Funktionscode in der schon vorhandenen `CASE`-Anweisung ab und legen Sie ein Unterprogramm `LOAD_TAB` an:

```
WHEN 'IMP'.
  PERFORM LOAD_TAB.
```

### Übung 3: Methoden des Controls aufrufen

5. Um den Text der internen Tabelle in den Editor zu laden, rufen Sie die Methode `set_text_as_r3table` in dem Unterprogramm auf. Mit dem Flag `g_loaded` merken Sie sich, daß der Text geladen wurde:

```
FORM LOAD_TAB.
  call method editor->set_text_as_r3table
    exporting table = mytable
  exceptions
    others = 1.
  if sy-subrc ne 0.
    CALL FUNCTION 'POPUP_TO_INFORM'
      EXPORTING
        TITEL = repid
        TXT2 = ''
        TXT1 = 'Error in set_text_as_r3table'(600).
  else.
    g_loaded = 'X'.
  endif.
  CALL METHOD CL_GUI_CFW=>FLUSH.
  IF SY-SUBRC NE 0.
    CALL FUNCTION 'POPUP_TO_INFORM'
      EXPORTING
        TITEL = REPID
        TXT2 = sy-subrc
        TXT1 = 'Form LOAD_TAB: Error in FLUSH'(601).
  ENDIF.
ENDFORM.          " LOAD_TAB
```

6. Aktivieren Sie die neuen Objekte und starten Sie Ihr Programm.

### Überprüfen Sie Ihre Arbeit

Wenn Sie auf die angelegte Drucktaste **Import** klicken, erscheint 20 mal die Zeile mit dem Text des Textelement `TEXT-001`. Wenn nicht, haben Sie wahrscheinlich vergessen das Textelement anzulegen.



In dieser Übung haben Sie eine Control-Methode innerhalb eines Unterprogramms aufgerufen. Eine Diskussion zu diesem Thema finden sie in [Übung 3: Control Methoden in Unterprogrammen \[Seite 48\]](#).

---

## Zusammenfassung

### Zusammenfassung

In dieser Lektion haben Sie gelernt, daß Sie Controls über eine bestimmte Klasse von Controls - den *Container* Controls - auf einem Dynpro integrieren. Container Controls verwalten ein oder mehrere Control-Instanzen und sind das Bindeglied zwischen Control und Dynpro.

Je nach Art des Control Containers müssen Sie einen Bereich im Screen Painter definieren, in den der Container plziert werden soll. Um eine Instanz eines Controls zu erzeugen, müssen Sie mit der Methode **CREATE OBJECT** ein Objekt der Klasse des Controls instanzieren.

Alle Methoden, die ein Control ansteuern, werden per RFC an das Frontend übertragen. Den Zeitpunkt, zu dem dies geschieht, bestimmen Sie selbst mit der Methode **FLUSH**. Dabei müssen Sie sicherstellen, daß zum Flush-Zeitpunkt noch alle Aktualparameter gültig sind. Systemfelder dürfen nicht übergeben werden, da sich diese während der Laufzeit oft ändern.

## Lektion 2: Ereignisbehandlung

### Aufgabenstellung

Sie haben Ihren Editor um die Funktion erweitert, den Inhalt einer internen Tabelle in das Textfenster zu laden. Nun soll ein Doppelklick auf eine Textzeile bewirken, daß die geklickte Zeile in eine Kommentarzeile umgewandelt wird.

### Überblick : Lerninhalte

Sie lernen in dieser Lektion:

- Wie Sie bei einem Control Ereignisse anmelden, die Sie abfangen möchten.
- Wie abgefangene Ereignisse im ABAP Programm Ereignisbehandlungsmethoden zugewiesen werden.
- Wie Sie Ereignisbehandlungsmethoden schreiben und diese dem Control bekannt machen.

### Grundlagen

Die Ereignisbehandlung von Controls baut auf die von *ABAP Objects* auf. Generell setzen Sie Ereignisse in ABAP Objects ein, um Objekte über das Eintreten eines Zustands zu informieren.

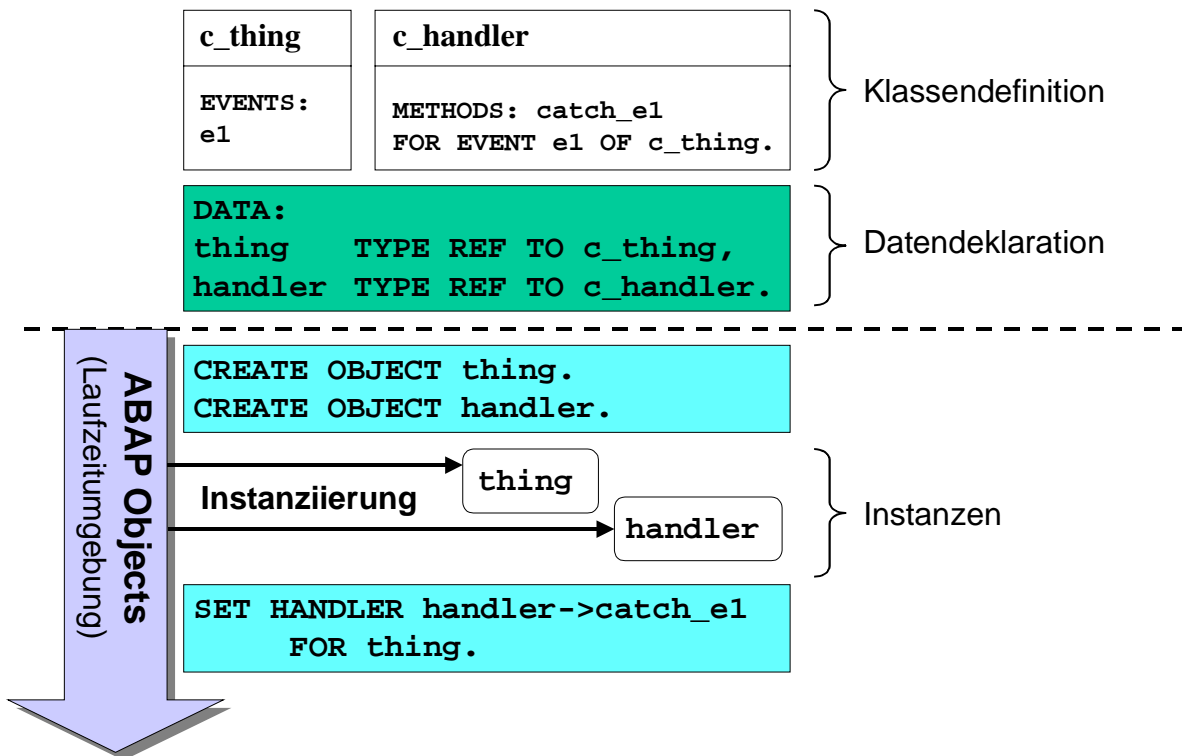


Mit einem Ereignis kann ein Objekt zum Beispiel folgendes signalisieren:

- Der Benutzer hat mit der Maus auf ein Bild geklickt.
- Der Benutzer hat in einem Web-Browser auf eine neue Seite navigiert.
- Das Objekt ist kurz davor, eine Tabelle auf dem Bildschirm auszugeben.

Das folgende Bild zeigt, wie Sie Ereignisse in Ihren Programmen nutzen können:

## Lektion 2: Ereignisbehandlung



Ein Ereignis ist eine Komponente einer Klasse oder eines Interface. Innerhalb der Methoden der Klasse können definierte Ereignisse mit **RAISE EVENT** ausgelöst werden.

Um auf ein Ereignis reagieren zu können, müssen Sie eine Methode definieren, die mit dem Ereignis verknüpft wird. Für die Behandlung des Ereignisses reicht es dann, dieses auszulösen. Solche Methoden werden auch *Ereignisbehandlungsmethoden* genannt. Durch die Definition einer zweiten Klasse kann man diese Methoden von dem Objekt trennen, das das Ereignis auslöst. Außerdem kann man so Behandlungsmethoden eines Typs in einer Klasse bündeln.



Im Bild wurde die Ereignisbehandlungsmethode als Instanzen-Methode definiert. Sie müssen dann erst ein Objekt der Klasse instanziiieren, bevor das System die Methode aufrufen kann. Es ist auch möglich, die Methode als statische Methode der Klasse zu definieren. In diesem Fall brauchen Sie das Objekt `handler` nicht erzeugen.

Nach der Instanziierung der Objekte verknüpfen Sie über die Anweisung **SET HANDLER** die Methode zur Ereignisbehandlung mit dem Objekt, in dem das Ereignis definiert ist. Die Laufzeitumgebung sorgt dann dafür, daß zu diesem Ereignis die implementierte Methode automatisch aufgerufen wird.

Weiterführende Informationen über die Ereignisbehandlung finden Sie in der Dokumentation von [ABAP Objects \[Extern\]](#).

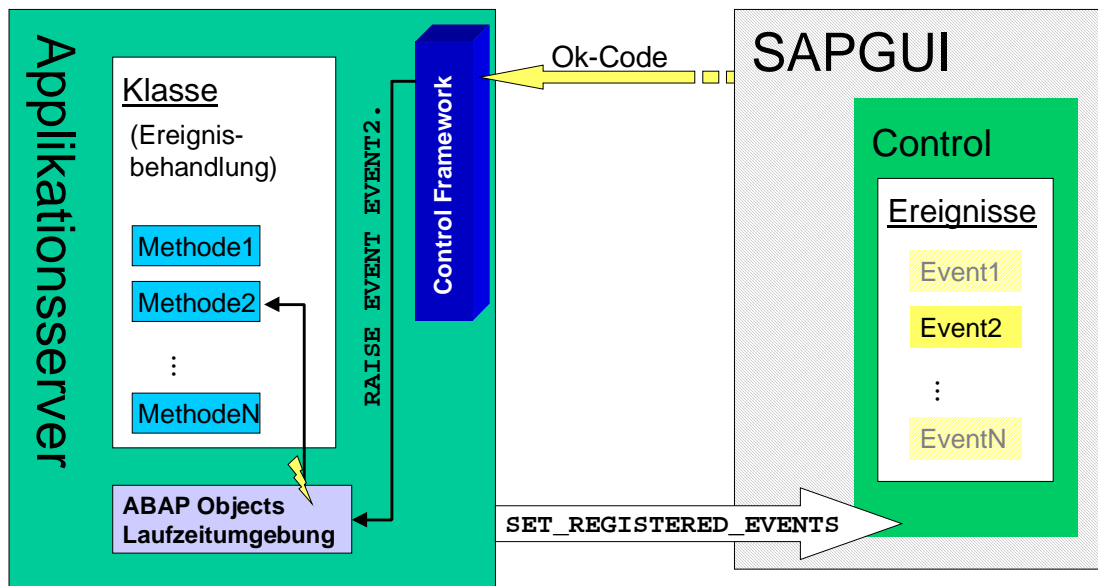
## Fortsetzung

[Lektion 3: Fluschoptimierung \[Seite 38\]](#) beschreibt die Verwendung des Flush-Aufrufs und wie die Methoden vor ihrer Übertragung zum Frontend gepuffert werden.

## Einführung

## Einführung

## R/3-Architektur



## Erläuterung

Das SAPGUI filtert im Initialzustand des Controls alle Ereignisse aus, die durch Benutzeraktionen vom Control ausgelöst werden. So wird verhindert, daß permanente Ereignisse – wie zum Beispiel Mausbewegungen – die Verbindung zum Applikationsserver zu stark belasten. Eine Behandlung von Ereignissen ist erst möglich, wenn Sie über Methoden der angelegten Instanz ausgewählte Ereignisse *registrieren*. Als Standardmethode steht hierfür die Methode `SET_REGISTERED_EVENTS` zur Verfügung.



Welche Ereignisse auf dem Frontend abgefangen werden können, entnehmen Sie der Klassendefinition des jeweiligen Control im [Class Builder \[Extern\]](#).

Das SAPGUI kodiert ein ausgelöstes Ereignis im Ok-Code, der vom Frontend zum Applikationsserver übertragen wird. So ein Ereignis, das auf dem Frontend ausgelöst wird, hat also noch nichts mit Ereignissen von ABAP Objects zu tun. Um dessen Ereignisbehandlung anzustoßen, muß das Control Framework noch das zugehörige Ereignis der entsprechenden Instanz zuordnen und auslösen.

Es gibt nun zwei Möglichkeiten, das ausgelöste Ereignis im ABAP Programm zu verarbeiten:

- Das Control Framework löst das Ereignis unabhängig von der Ablauflogik des Dynpros aus, d.h. ohne die Ereignisse PAI und PBO zu prozessieren. Solche Ereignisse werden *System-Ereignisse* genannt. Diese Art der Ereignisverarbeitung ist als default eingestellt.

- Das Control Framework prozessiert nach einem Ereignis den PAI. In diesem Fall müssen Sie die Methode `CL_GUI_CFW=>DISPATCH` aufrufen, um die Ereignisbehandlung von ABAP Objects selbst anzustoßen. Solche Ereignisse werden *Anwendungs-Ereignisse* genannt.



Sie müssen selbst entscheiden, ob Sie für Ihre Anwendung System-Ereignisse oder Anwendungs-Ereignisse benötigen. Die Übungen [Übung 2: Ein Anwendungs-Ereignis registrieren \[Seite 30\]](#) und [Übung 3: Ein System-Ereignis registrieren \[Seite 32\]](#) diskutieren die Konsequenzen der jeweiligen Entscheidung.

Schließlich wird die Methode, die Sie mit dem Ereignis verknüpft haben, automatisch aufgerufen.

## Folgerungen

Um Ereignisse, die von einem Control ausgelöst werden, verarbeiten zu können, müssen Sie:

- Die Ereignisse bei der Control-Instanz registrieren, so daß sie nicht mehr vom SAPGUI zurückgehalten werden.
- Bei Anwendungs-Ereignissen müssen Sie die statische Methode `DISPATCH` im PAI aufrufen. Dadurch werden die übermittelten Ereignisse auseinandergesteuert und entsprechende Ereignisbehandlungsmethoden - sofern vorhanden - aufgerufen.
- Eine Klasse mit Ereignisbehandlungsmethoden definieren. In den Methoden können Sie weitere Informationen zum Ereignis abfragen und das weitere Verhalten des Control steuern.
- Die Ereignisse Ihrer Control-Instanz mit der jeweiligen Ereignisbehandlungsmethode verknüpfen. Sie benutzen dafür die Methode `SET HANDLER`.

## Übung 1: Eine Ereignisbehandlungsmethode schreiben

# Übung 1: Eine Ereignisbehandlungsmethode schreiben

## Verwendung

Ereignisse eines Controls, die der Benutzer am Frontend auslöst, werden am Backend über die Ereignisverarbeitung von ABAP Objects verarbeitet. Dazu müssen Sie für jedes Ereignis, auf das Sie reagieren möchten, eine Ereignisbehandlungsmethode implementieren.

## Vorgehensweise

1. Definieren Sie eine lokale Klasse vor dem Ereignis **START-OF-SELECTION**. Implementieren Sie für diese Klasse eine Methode, die beim Ereignis **DBLCLICK** aufgerufen wird. Die Methode gibt lediglich den Ereignistyp über ein Textelement aus:

```
DATA: EVENT_TYPE(20) TYPE C.
CLASS lcl_event_handler DEFINITION.
  PUBLIC SECTION.
    CLASS-METHODS: CATCH_DBLCLICK FOR EVENT DBLCLICK OF CL_GUI_TEXTEDIT.
                    IMPORTING SENDER.
ENDCLASS.

DATA: event_handler TYPE REF TO lcl_event_handler.

CLASS lcl_event_handler IMPLEMENTATION.
  METHOD CATCH_DBLCLICK.
    event_type = text-002.
  ENDMETHOD.
ENDCLASS.
```

Vergessen Sie nicht, für das Textelement **text-002** einen Text, zum Beispiel, **Doppelklick** zu vergeben.

2. Legen Sie für die Ausgabe des Ereignistyps ein Ein-/Ausgabefeld auf Dynpro 100 an:
  - Name: **EVENT\_TYPE**
  - Elementtyp: Ein/Ausgabefeld
3. Verknüpfen Sie die Methode mit dem Ereignis. Fügen Sie dazu den folgenden Code in den IF-Block des PBO-Moduls ein (das Objekt **editor** muß schon erzeugt sein):

```
SET HANDLER lcl_event_handler=>catch_dblclick FOR editor.
```

4. Aktivieren und starten Sie Ihr Programm.

## Überprüfen Sie Ihre Arbeit

Das Ein-/Ausgabefeld wird bei einem Doppelklick innerhalb des Textfensters nicht aktualisiert. Das ABAP Objects Ereignis wird also noch nicht ausgelöst.

## Diskussion

Die Schritte in dieser Übung spiegeln nur die Behandlung eines Ereignisses unter ABAP Objects wider. Die Ereignisbehandlung am Backend ist also beschrieben. Da alle Ereignisse im Initialzustand des Controls vom SAPGUI ausgefiltert werden, kann das ABAP-Ereignis zum Doppelklick aber nicht am Backend ausgelöst werden. Grund hierfür ist die noch fehlende

## Übung 1: Eine Ereignisbehandlungsmethode schreiben

Registrierung des Ereignisses am Frontend (siehe [Übung 2: Ein Anwendungs-Ereignis registrieren \[Seite 30\]](#)).

### Ereignisparameter `sender`

In der Definition der Ereignisbehandlungsmethode wird der Ereignisparameter `sender` importiert. Auf diesen Parameter können Sie bei jedem ABAP Objects Ereignis zugreifen, obwohl er im Class Builder nicht als Ereignisparameter aufgeführt ist. Der Parameter importiert eine Referenz auf die Instanz, die das Ereignis ausgelöst hat. Die Referenz kann direkt dazu genutzt werden, um Methoden dieser Instanz aufzurufen.

## Die Ereignisbehandlungsmethode

### Definition als statische Methode

In dieser Übung wurde die Ereignisbehandlungsmethode mit dem Zusatz `CLASS` als statische Methode der Klasse `CL_GUI_TEXTEDIT` definiert. Sie brauchen also kein Objekt der Klasse `lcl_event_handler` instanziiieren, um die Methode aufrufen zu können.

Mit `SET HANDLER` verknüpfen Sie die statische Methode `lcl_event_handler=>catch_dblick` mit dem Ereignis `DBLCLICK` für die Instanz `editor`. Sie könnten auch noch weitere Instanzen des Textedit Controls erzeugen und mit dieser Methode bei diesem Ereignis verknüpfen. Alle mit `SET HANDLER` angemeldeten Instanzen benutzen dann die gleiche Methode, die nur einmal zur Laufzeit existiert.



Der Ereignisparameter `sender` enthält eine Referenz auf die auslösende Instanz, über die Sie Methoden dieser Instanz aufrufen können.

Die Definition der Ereignisbehandlungsmethode als statische Methode ist also dann sinnvoll, wenn sich alle Instanzen einer Klasse bei einem Ereignis gleich verhalten sollen.

### Definition als Instanzen-Methode

Vorteil bei Instanzen-Methoden ist, daß Sie mehrere Objekte Ihrer lokalen Klasse erzeugen können. Dies ist nötig, wenn Sie sich innerhalb der Ereignisbehandlungsmethode einen Zustand merken möchten (z.B. über eine statische Variable). Arbeiten mehrere Control-Instanzen mit der Methode, kann es bei nur einer Instanz der Ereignisbehandlungsklasse zu Konflikten kommen. In einem solchen Fall sollten Sie verschiedene Instanzen-Methoden für verschiedene Control-Instanzen der gleichen Klasse verwenden.

## Übung 2: Ein Anwendungs-Ereignis registrieren

# Übung 2: Ein Anwendungs-Ereignis registrieren

## Verwendung

Um auf Ereignisse reagieren zu können, müssen Sie erst bei der Control-Instanz und damit am Frontend registriert werden. Im Initialzustand des Control filtert das SAPGUI alle Ereignisse aus.

In dieser Übung registrieren Sie das Ereignis `DBLCLICK` beim Textedit Control als Anwendungs-Ereignis. Zunächst soll nur auf dem Dynpro ein Text zum ausgelösten Ereignis angezeigt werden.

## Vorgehensweise

1. Deklarieren Sie folgende Variablen im Rahmenprogramm: a) Eine interne Tabelle für die Ereignisse, die Sie registrieren möchten und b) eine Struktur für eine Zeile dieser Tabelle:

```
DATA events TYPE cntl_simple_events.
DATA wa_events TYPE cntl_simple_event.
```

Fügen Sie die nächsten Code-Blöcke im PBO nach der Erzeugung des Control innerhalb des IF-Blocks ein.

2. Weisen Sie dem Feld `eventid` das statische Attribut für das Ereignis `DBLCLICK` zu:

```
wa_events-eventid = cl_gui_textedit=>event_double_click.
```

3. Bestimmen Sie mit Hilfe des Feldes `appl_event`, daß der Typ des Ereignisses ein Anwendungs-Ereignis ist:

```
wa_events-appl_event = 'X'.
```

4. Hängen Sie die Zeile an Ihre interne Tabelle `events` an und übergeben Sie diese Tabelle an das Textedit-Control mit der Methode `set_registered_events`:

```
APPEND wa_events TO events.
CALL METHOD editor->set_registered_events
EXPORTING events = events.
```



Diese Methode stellen alle Controls zur Verfügung. Außerdem kann es je nach Control zusätzliche Methoden geben, um ein einzelnes Ereignis zu registrieren.

5. Rufen Sie im PAI die Methode `DISPATCH` in der letzten Abfrage der CASE-Anweisung auf:

```
WHEN OTHERS.
CALL METHOD CL_GUI_CFW=>DISPATCH.
```

6. Aktivieren und starten Sie Ihr Programm.

## Überprüfen Sie Ihre Arbeit

Wenn Sie in dem Textfenster des Editors doppelklicken, erscheint der Text des Textelements `text-002` auf dem Dynpro.

## Übung 2: Ein Anwendungs-Ereignis registrieren

### Diskussion

Die Registrierung des Ereignisses `DBLCLICK` bewirkt, dass das Frontend Doppelklicks des Benutzers über den OK-Code an das Backend übermittelt. Da Sie es als Anwendungs-Ereignis registriert haben, prozessiert das Control Framework den PAI des betroffenen Dynpros. Mit der Methode `DISPATCH` legen Sie dann fest, wann das Ereignis von ABAP Objects ausgelöst wird. Die Laufzeitumgebung führt dann die Methode `CATCH_DBLCLICK` aus und setzt die Abarbeitung nach dem `DISPATCH`-Aufruf fort.



Der `DISPATCH`-Aufruf gilt global für alle angelegten Control-Instanzen des Rahmenprogramms.

### Vorteile

- Sie bestimmen selbst den Zeitpunkt, wann die Ereignisbehandlungsmethode angesprochen werden soll. Dazu verwenden Sie die Methode `DISPATCH`.
- Da das System den PAI prozessiert, hat das System den Feldtransport zwischen Dynpro und Applikationsserver bereits durchgeführt. Sie können also in der Ereignisbehandlungsmethode auf Dynprofelder zugreifen.

### Nachteile

Allerdings gibt es bei der Verwendung von Anwendungs-Ereignissen Probleme bei der Durchführung von Eingabepflichten mit Hilfe der `FIELD- beziehungsweise CHAIN`-Anweisung. Bei fehlerhaften Eingaben kann der Benutzer gezwungen werden, seine Eingaben zu wiederholen. Dabei kann er aber auch weitere Ereignisse auf dem Control auslösen. Auf diese Weise können also Ereignisse verloren gehen.



Die mehrfache Verwendung der Methode `DISPATCH` ist auch keine Lösung, da die Ereignisbehandlungsmethoden darauf ausgelegt sind, einmal aufgerufen zu werden.

## Übung 3: Ein System-Ereignis registrieren

# Übung 3: Ein System-Ereignis registrieren

## Verwendung

System-Ereignisse werden unabhängig von der Ablauflogik des entsprechenden Dynpros weitergeleitet.

In der letzten Übung haben Sie das Ereignis **DLBCLICK** als Anwendungs-Ereignis registriert. Registrieren Sie es nun als System-Ereignis.

## Vorgehensweise

1. Ändern Sie den Wert des Feldes `wa_events-appl_event` im PBO:  
`wa_events-appl_event = SPACE.`
2. Kommentieren Sie den Aufruf der Methode **DISPATCH** im PAI aus.
3. Aktivieren und starten Sie Ihr Programm.

## Überprüfen Sie Ihre Arbeit

Doppelklicken Sie in das Textfenster. Es erscheint kein Text mehr, weil die Ablauflogik des Dynpros und der Feldtransport umgangen wird.

## Diskussion

Bei System-Ereignissen wird kein PAI/PBO prozessiert. Daraus ergeben sich folgende Konsequenzen:

### Vorteile

- Die Methode **DISPATCH** wird automatisch aufgerufen.
- Ereignisse können nicht verloren gehen, weil die Ablauflogik des Dynpros nicht abgearbeitet wird. Feldprüfungen und damit verbundene Neueingaben auf dem Dynpro fallen weg und kollidieren nicht mit der Ereignisbehandlung.

### Nachteile

Es findet beim Auslösen des Ereignisses kein Feldtransport statt. Die Folge für Zugriffe auf Dynprofelder innerhalb von Ereignisbehandlungsmethoden:

- Lesende Zugriffe greifen auf veraltete Werte zu, genauer: auf die Werte des letzten Feldtransports.
- Schreibende Zugriffe gehen bei Ein-/Ausgabefeldern ganz verloren. Das System überschreibt beim folgenden Feldtransport die Zuweisung mit dem Inhalt des Dynprofeldes.



Dieses Problem tritt bei reinen Ausgabefeldern nicht auf.

Solange Sie in einer Ereignisbehandlungsmethode auf keine Dynprofelder lesend oder schreibend zugreifen, hat dieser Nachteil also keine Auswirkungen.

---

**Übung 3: Ein System-Ereignis registrieren****Abhilfe**

Das Control Framework stellt die Methode `SET_NEW_OK_CODE` zur Verfügung. Mit dieser Methode können Sie innerhalb einer Ereignisbehandlungsmethode einen anwendungsspezifischen `OK_Code` setzen. Das Control Framework prozessiert dann nach der Ereignisbehandlungsmethode das PAI-Ereignis, in dem Sie wieder aktuelle Dynprofelder lesen und schreiben können.

Ändern Sie Ihr Programm folgendermaßen:

1. Kommentieren Sie in der Ereignisbehandlungsmethode die Zeile `event_type = text-002.` aus.

2. Setzen Sie in der Ereignisbehandlungsmethode den anwendungsspezifischen Ok-Code `SHOW`:

```
CALL METHOD cl_gui_cfw=>set_new_ok_code  
  exporting new_code = 'SHOW'.
```

3. Fragen Sie den Ok-Code im PAI ab. Ergänzen Sie dazu Ihre Case-Abfrage:

```
when 'SHOW'.  
  event_type = 'Doubleclick'(555).
```

4. Aktivieren und starten Sie Ihr Programm.

Das Ereignis PAI wird nun nach der Ereignisbehandlungsmethode prozessiert und der Text wieder ausgegeben.

## Übung 4: Auf ein Ereignis reagieren

# Übung 4: Auf ein Ereignis reagieren

## Verwendung

In der letzten Übung haben Sie ein Ereignis registriert. Die Ereignisbehandlungsmethode wurde dabei über die Anweisung `SET HANDLER` mit dem Ereignis verknüpft. Um auf das Ereignis zu reagieren, brauchen sie nur noch die Ereignisbehandlungsmethode erweitern.

In dieser Übung implementieren Sie eine Funktion, die eine per Doppelklick ausgewählte Zeile zu einer Kommentarzeile macht. Ist die Zeile schon eine Kommentarzeile, so wird die Kommentar-Markierung am Beginn der Zeile gelöscht. Realisieren Sie diese Funktion, indem Sie in der Ereignisbehandlungsmethode weitere Informationen vom Control abfragen und über Methoden Aktionen beim Control auslösen.

## Vorgehensweise

Erweitern Sie die Methode `catch_dbclick` vor dem Aufruf von `set_new_ok_code` um folgende Codebausteine:

1. Fragen Sie die Position des Doppelklicks ab und deklarieren Sie vorher entsprechend lokale Variablen:

```
DATA: FROM_LINE TYPE I,
      FROM_POS TYPE I,
      TO_LINE TYPE I,
      TO_POS TYPE I.
CALL METHOD SENDER->GET_SELECTION_POS
IMPORTING
  FROM_LINE = FROM_LINE
  FROM_POS = FROM_POS
  TO_LINE = TO_LINE
  TO_POS = TO_POS.
```

2. Da sich der Text im Textbuffer des Editors geändert haben könnte, laden sie ihn zunächst wieder in Ihre interne Tabelle `mytable`. Mit dem Flag `g_loaded` Stellen Sie fest, ob überhaupt schon Text in den Textbuffer geladen wurde:

```
IF NOT g_loaded IS INITIAL.
  CALL METHOD SENDER->GET_TEXT_AS_R3TABLE
  importing
    TABLE = MYTABLE.
ENDIF.
```

3. Lesen Sie die Zeile der internen Tabelle, auf die geklickt wurde. Je nachdem, ob die Zeile schon als Kommentarzeile markiert ist, fügen Sie einen Stern ein oder löschen ihn wieder:

```
READ TABLE MYTABLE INDEX FROM_LINE INTO TEXTSTRUCT.
IF SY-SUBRC = 0.
  IF TEXTSTRUCT+0(1) CS '*'.
    SHIFT TEXTSTRUCT.
  ELSEIF TEXTSTRUCT+0(1) NS '*'.
    SHIFT TEXTSTRUCT RIGHT.
    TEXTSTRUCT+0(1) = '*'.
  ENDIF.
```

## Übung 4: Auf ein Ereignis reagieren

4. Laden Sie ihren Editor wieder mit dem Text der internen Tabelle.

```
MODIFY MYTABLE FROM TEXTSTRUCT INDEX FROM_LINE.
  CALL METHOD SENDER->SET_TEXT_AS_R3TABLE
    EXPORTING TABLE = MYTABLE
  EXCEPTIONS
    OTHERS = 1.
ENDIF.
```

5. Übertragen Sie die Methoden mit dem Befehl **FLUSH** zum Frontend.

```
CALL METHOD cl_gui_cfw=>FLUSH.
IF SY-SUBRC NE 0.
  CALL FUNCTION 'POPUP_TO_INFORM'
    EXPORTING
      TITEL = REPID
      TXT2 = sy-subrc
      TXT1 = 'Method CATCH_DBLCLICK: Error in Flush!(602).
ENDIF.
```

6. Aktivieren Sie Ihr Programm und starten Sie es.

## Überprüfen Sie Ihre Arbeit

Um den generierten Text der internen Tabelle im Textfenster anzuzeigen, klicken Sie auf die Drucktaste `Import`. Ein Doppelklick in einer Zeile bewirkt nun, daß ein Stern (\*) am Beginn der Zeile erscheint. Der Text wird dabei nach rechts eingerückt. Ein weiterer Doppelklick auf die gleiche Zeile bewirkt, daß der Stern wieder entfernt wird.

## Diskussion

Die Methode `get_text_as_r3table` lädt den Text im Textfenster des Control in die interne Tabelle `mytable`. Im nächsten Schritt wird diese Tabelle modifiziert. Zeitlich gesehen, sollte die Methode allerdings erst zum Zeitpunkt des Flush-Aufrufs ausgeführt werden. Die Änderung wäre also demnach zu früh. Warum verhält sich das Programm aber trotzdem korrekt?

Der Grund ist, daß bei großen Datenmengen, die von Control-Methoden übertragen werden, ein weiteres Control genutzt wird: Der Data Provider. Dieser löst zur Zeit noch implizit einen weiteren Flush aus, um die Daten zu übertragen.



Einen impliziten Flush können Sie im Trace Modus aufspüren. Dieser Modus wird in der vierten Lektion in [Übung 1: Den Trace Modus aktivieren \[Seite 56\]](#) beschrieben.

Es schadet aber nichts, nach der Methode `get_text_as_r3table` einen weiteren Flush einzubauen. Wird ein Flush aufgerufen, ohne daß vorher Control-Methoden aufgerufen werden, so führt das Control Framework keinen RFC aus.



Es wird empfohlen, diesen zusätzlichen Flush-Aufruf zu verwenden statt auf den impliziten Flush zu vertrauen. Dies trägt zur Robustheit bei, denn fällt in späteren Versionen des Data Provider der implizite Flush weg, sind die bis dahin geschriebenen Programme weiterhin korrekt.

Die nächste Lektion geht ausführlich auf die richtige Verwendung von Flush-Aufrufen ein.

Übung 4: Auf ein Ereignis reagieren

## Zusammenfassung

In Lektion 2 haben Sie gelernt, wie Sie Ereignisse eines Control abfangen und darauf reagieren. Im Wesentlichen müssen Sie dafür folgendes tun:

- Das Ereignis beim SAPGUI registrieren. Sie verwenden dazu die Methode `set_registered_events`.
- Bei der Registrierung entscheiden, ob das Ereignis als *System-Ereignis* oder als *Anwendungs-Ereignis* registriert werden soll.
- Eine Ereignisbehandlungsmethode definieren. Dazu können Sie zum Beispiel eine lokale Klasse in Ihrem Programm definieren.
- Das Ereignis mit der Ereignisbehandlungsmethode verknüpfen. Dazu verwenden Sie die Anweisung `SET HANDLER`.
- Im Falle von Anwendungs-Ereignissen die statische Methode `c1_gui_cfw=>DISPATCH` im PAI aufrufen.

---

**Lektion 3: Flushoptimierung**

## Lektion 3: Flushoptimierung

### Aufgabenstellung

In dieser Lektion erweitern Sie den Texteditor um die Funktion, Textzeilen im Editorfenster gegen Änderungen zu schützen.

### Überblick: Lerninhalte

Eine Optimierung der Anzahl von Flush-Aufrufen heißt in Kurzfassung: Bauen Sie so wenig wie möglich in Ihre Programme ein. Das bedeutet aber, daß Sie wissen müssen, an welchen Stellen eine Synchronisation in Ihrem Programm zwingend notwendig ist.

In den Übungen wird deshalb zunächst die jeweils 'naive' Lösung vorgestellt. Danach wird diese dann bezüglich der Flushproblematik diskutiert und Verbesserungsvorschläge gemacht. Ist man sich der Effekte bei der Anwendung der Regel "sowenig Flush-Aufrufe wie möglich" bewußt, kann man deren Anzahl reduzieren.

Sie lernen in dieser Lektion:

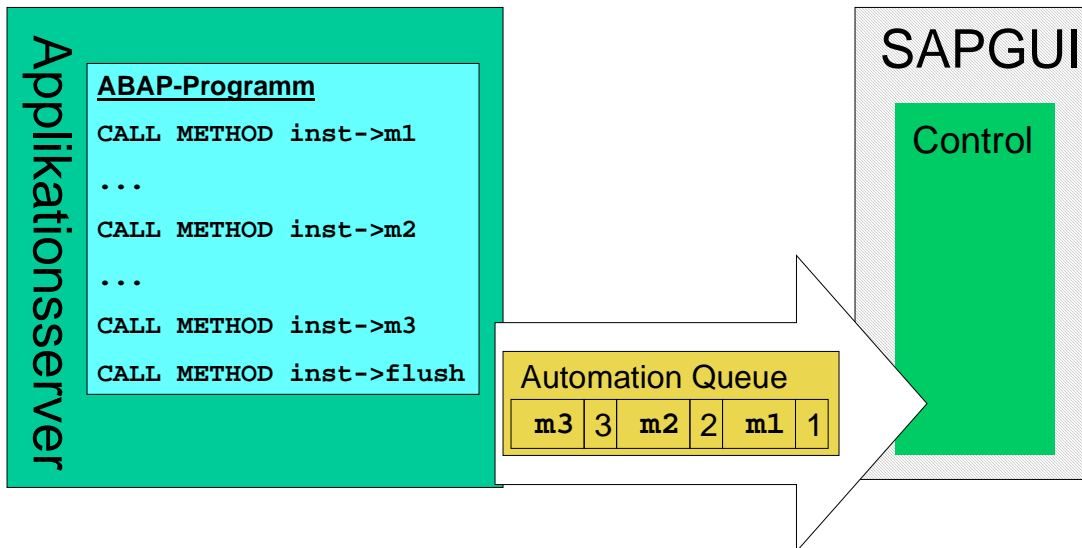
- Wann es zu Fehlern bei falscher Anwendung von Flush-Aufrufen kommen kann.
- Wie Sie die Anzahl von Flush-Aufrufen reduzieren.
- Was implizit mit Aktualparametern von Control-Methoden gemacht wird.

### Fortsetzung

[Lektion 4: Testen und Fehlerbehandlung \[Seite 53\]](#) beschreibt, was bei der Fehlersuche in Control Programmen zu beachten ist.

## Einführung

### R/3-Architektur



### Erläuterung

Die Control-Verschaltung stellt an Ihrer Schnittstelle Methoden zur Steuerung des Control bereit. Der Aufruf einer solchen Methode impliziert aber noch nicht dessen Ausführung zur Laufzeit. Das System puffert diese Methoden zunächst in einer Warteschlange, der *Automation Queue*, und zwar zum Aufrufzeitpunkt im ABAP-Programm. Die Reihenfolge bei der Ausführung dieser Methoden bleibt somit erhalten. Ausgeführt werden die Methoden allerdings erst, wenn sie mit der Methode **FLUSH** per RFC zum Frontend übertragen werden. Über die Automation Queue wird also die Anzahl der nötigen RFC-Aufrufe reduziert.

Durch das Puffern der Control-Methoden in die Automation Queue werden deren Aufrufe und das restliche ABAP-Coding bei Ausführung zeitlich entkoppelt. Die Methode **FLUSH** legt somit einen Synchronisationszeitpunkt fest.



Aus Optimierungsgründen kann es vorkommen, daß das Control Framework die Automation Queue nicht überträgt, obwohl Sie die Methode **FLUSH** aufgerufen haben. Dies macht zum Beispiel Sinn, wenn die Queue leer ist oder nur Methoden mit **EXPORTING**-Parametern enthält. Mehr hierzu finden Sie in [Verwendung von Controls im WAN \[Extern\]](#) der Control Framework Dokumentation.

## Einführung

### Art der Parameterübergabe

Entscheidend für eine robuste Programmierung von Controls ist das Verständnis der Automation Queue. Wird eine Methode in die Queue aufgenommen, so gibt es zwei Möglichkeiten, wie das Control Framework die Parameter der Methode speichert:

- Die Werte der Aktualparameter werden in die Queue gestellt (call-by-value).
- Die Referenzen auf Aktualparameter werden in die Queue gestellt (call-by-reference).

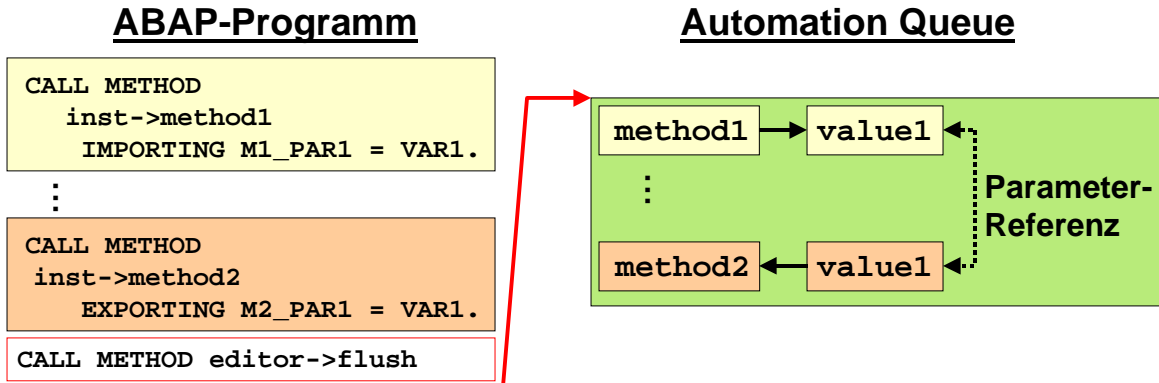


Sie haben keinen Einfluß auf die Entscheidung, wie die Aktualparameter gepuffert werden. Sie übergeben die meisten Aktualparameter von Control-Methoden per Referenz. Das Control Framework entscheidet dann auf Grund von Optimierungsaspekten über die Art der Pufferung. Aktualparameter, die an der Methodenschnittstelle als Werteparameter deklariert sind, werden immer als Wert übergeben.

### Call-by-Reference

Das Framework speichert die Referenz auf einen Aktualparameter in der Queue, wenn die Variable beim Methodenaufwurf importiert, d.h., an das Programm zurückgegeben wurde. Folgen weitere Methoden mit dem gleichen Parameter in der Queue, so werden diese ebenfalls per Referenz gespeichert.

Gleiche Referenzen innerhalb der Queue werden vom Framework erkannt und über eine *Parameter-Referenz* verbunden:



Das heißt konkret: Werte von Variablen, die im Programm bei einem Methodenaufwurf importiert wurden, können bei folgenden Methodenaufrufen innerhalb der Queue exportiert werden. Eine Rückgabe der Kontrolle zum Backend ist also nicht nötig. Dank dieses Mechanismus können RFC-Aufrufe eingespart werden, so daß sich die Performance erhöht.

### Call-by-Value

In allen anderen Fällen als oben beschrieben speichert das Control Framework **die Werte** der Aktualparameter in der Automation Queue. Es steht also schon vor dem Flush-Aufruf fest, welcher Wert übergeben wird.

## Folgerungen

Ein RFC-Aufruf stellt einen Flaschenhals in der Verbindung zum Frontend dar. Deshalb sollten Sie in Ihren Programmen die Anzahl der Flush-Aufrufe soweit möglich reduzieren.

Durch die zeitliche Entkopplung der Methodenaufrufe und des restlichen Code kann es aber bei falscher Synchronisation zu folgenden Effekten kommen:

- Die Werte von importierten Variablen sind nicht aktuell (wirkt sich auf Bedingungs-Abfragen aus) oder werden zu früh geändert (die Änderungen werden dann überschrieben). Dies tritt auf, wenn die Automation Queue zu spät durch einen Flush-Aufruf abgearbeitet wird.
- Die Änderung einer Variable wird zwischen zwei Methodenaufrufen unwirksam, wenn sie von der ersten Methode importiert und der zweiten exportiert wird. Dies tritt auf, weil die Variable innerhalb der Automation Queue übergeben wird.
- Aktualparameter, die als Referenz übergeben wurden, sind nicht mehr gültig. Dies tritt auf, wenn die Methode in einem Unterprogramm aufgerufen wurde, aber erst nach dessen Ende die Methode `FLUSH` aufgerufen wird.

Die Übungen dieser Lektion dienen als Diskussionsgrundlage für die Flushproblematik.

## Übung 1: Importierte Werte verwenden

# Übung 1: Importierte Werte verwenden

## Verwendung

Bevor Sie auf Benutzeraktionen reagieren, fragen Sie üblicherweise den aktuellen Zustand des Control ab. Dabei verwenden Sie Control-Methoden, die Werte an Ihr Programm übergeben. Wenn Sie diese Werte verändern oder abfragen, müssen Sie sicherstellen, daß sie nach dem Methodenaufruf mit den aktuellen Werten arbeiten. Dies hängt vom richtigen Flushzeitpunkt ab.

In dieser Übung implementieren Sie eine Funktion, die dem Benutzer ein Schützen von selektierten Zeilen erlaubt. Dazu fragen Sie den selektierten Bereich mit der Methode `GET_SELECTION_POS` ab und schützen diesen dann mit der Methode `PROTECT_LINES`.

## Vorgehensweise

1. Legen Sie auf Ihrem Dynpro eine Drucktaste mit den folgenden Eigenschaften an:

- Funktionscode: `PROTECT`
- Icon-Name: `ICON_LOCKED`

In den nächsten Schritten erweitern Sie den `CASE`-Block im `PAI`-Modul:

2. Deklarieren Sie globale Variablen für den Selektionsbereich und einen Laufindex:

```
WHEN 'PROTECT'.
  DATA: FROM_IDX TYPE I,
        TO_IDX TYPE I,
        INDEX TYPE I.
```

3. Fragen Sie den den Bereich ab, der vom Benutzer mit der Maus selektiert wurde:

```
CALL METHOD EDITOR->GET_SELECTION_POS
  IMPORTING
    FROM_LINE = FROM_IDX
    TO_LINE = TO_IDX
exceptions
  ERROR_CNTL_CALL_METHOD = 1.
```

4. Synchronisieren Sie die Ausführung am Control mit dem ABAP Programm:

```
CALL METHOD cl_gui_cfw=>FLUSH.
IF SY-SUBRC NE 0.
  CALL FUNCTION 'POPUP_TO_INFORM'
  EXPORTING
    TITEL = REPID
    TXT2 = sy-subrc
    TXT1 = 'PAI USER_COMMAND_100(1): Error in Flush!(603).
ENDIF.
```



Ohne diese Synchronisation haben die Variablen `FROM_IDX` und `TO_IDX` in der folgenden `IF`-Abfrage veraltete Werte (initial beide 0).

## Übung 1: Importierte Werte verwenden

5. Schützen Sie die markierten Zeilen mit Hilfe einer Schleife. Sie stellen dabei pro Zeile einen Methodenaufruf in die Automation Queue, die Sie danach flushen:

```

LOOP AT MYTABLE INTO TEXTSTRUCT.
  IF ( SY-TABIX >= FROM_IDX AND SY-TABIX <= TO_IDX ).
    INDEX = SY-TABIX.
    CALL METHOD EDITOR->PROTECT_LINES
      EXPORTING
        FROM_LINE = INDEX
        TO_LINE   = INDEX
        ENABLE_EDITING_PROTECTED_TEXT = cl_gui_textedit=>true.
  endif.
ENDLOOP.
CALL METHOD cl_gui_cfw=>FLUSH.
IF SY-SUBRC NE 0.
  CALL FUNCTION 'POPUP_TO_INFORM'
    EXPORTING
      TITEL = REPID
      TXT2 = sy-subrc
      TXT1 = 'PAI USER_COMMAND_100(2): Error in Flush!'(604).
ENDIF.

```



Die Anwendung der Methode `PROTECT_LINES` ist recht naiv. Einfacher wäre es, nur einen Methodenaufruf zu verwenden, bei dem man `from_idx` und `to_idx` übergibt. Das Beispiel wurde so gewählt, um auf besondere Effekte bei der Control-Programmierung aufmerksam zu machen.

6. Aktivieren Sie Ihre neuen Objekte und starten Sie Ihr Programm.

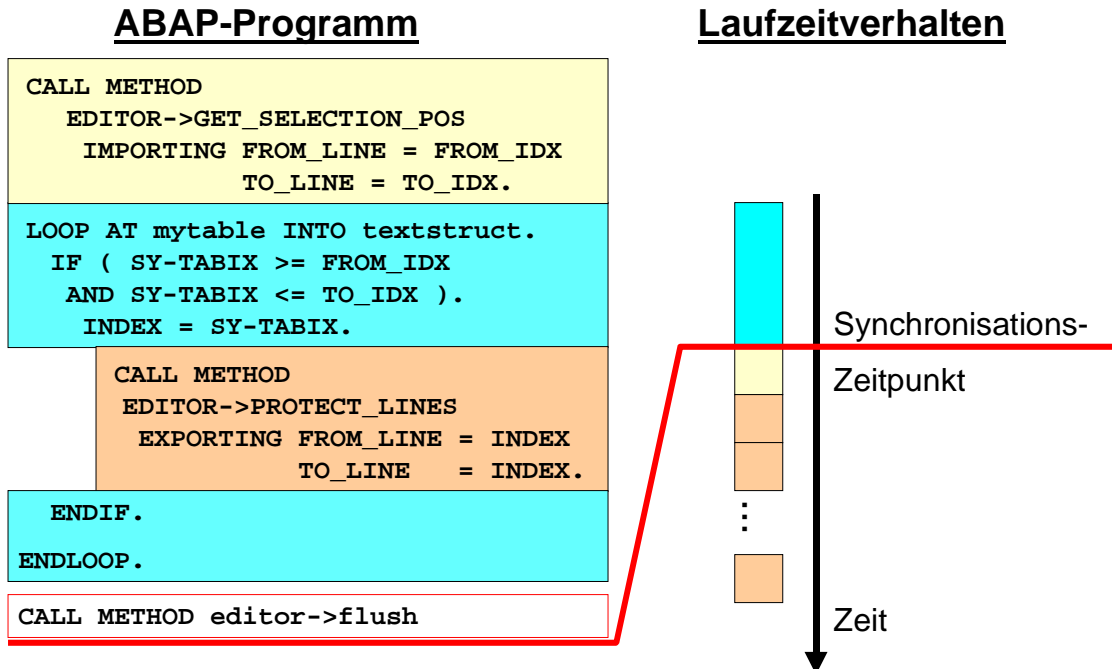
## Überprüfen Sie Ihre Arbeit

Importieren Sie über die schon angelegte Drucktaste Ihre interne Tabelle. Wenn Sie den Cursor auf eine Zeile positionieren und auf die neue Drucktaste klicken, wird die Zeile grau markiert. Sie ist dann nicht mehr änderbar. Mit der linken Maustaste können Sie einen ganzen Bereich markieren, um mehrere Zeilen gegen Eingaben zu schützen.

## Diskussion

Bei dieser Lösung ist die Synchronisation nach dem Methodenaufruf `GET_SELECTION_POS` zwingend notwendig. Kommentieren Sie die `FLUSH`-Anweisung testweise aus und verfolgen Sie im Debugger, welche Werte die Variablen `from_idx` und `to_idx` haben. Es ergibt sich während der Laufzeit folgendes Verhalten (aus Platzgründen wurde auf den Parameter `ENABLE_EDITING_PROTECTED_TEXT` der Methode `PROTECT_LINES` in der grafischen Darstellung verzichtet):

## Übung 1: Importierte Werte verwenden



Ohne **FLUSH** nach dem Aufruf von **GET\_SELECTION\_POS** werden die Werte von **from\_idx** und **to\_idx** zu früh überprüft. Beim ersten Durchlaufen des PAI sind beide Variablen initial. Wegen der Schleifenbedingung wird deshalb kein einziger Methodenaufruf **PROTECT\_LINES** in die Automation Queue gestellt. Wird die Funktion danach nochmal aufgerufen, so wird die Schleifenbedingung immer gegen zu alte Werte geprüft. Dann werden zwar Zeilen geschützt, aber gerade die vom letzten Aufruf.



Den zweiten Flush in dieser Übung kann man vermeiden, indem man auf die Methode **PROTECT\_SELECTION** verwendet. Auf diese Weise braucht man nicht mehr die Zeilennummern des markierten Bereichs abfragen.

### Werte in der Automation Queue

In diesem Beispiel gibt es keine Methode, die vor **PROTECT\_LINES** in der gleichen Queue aufgerufen wird und den Aktualparameter **INDEX** importiert. Deshalb puffert das Control Framework die Aktualparameter der Methoden als Werte in der Queue. Die nächste Übung gibt ein Beispiel, in dem das Framework Referenzen von Aktualparametern in der Automation Queue speichert.

## Übung 2: Wertübergabe zwischen Methoden

### Verwendung

Methodenparameter, die über Referenzen übergeben werden (call-by-reference), können innerhalb der Automation Queue weitergereicht werden. Somit können Methoden Rückgabewerte von vorausgehenden Methoden in der Queue verwenden.

Diese Übung konstruiert ein Beispiel, in dem diese Möglichkeit gerade unerwünscht ist. Sie implementieren eine Abfrage, die das Schützen der ersten sichtbaren Zeile im Textfenster verbietet. Der Benutzer soll bei einem derartigen Versuch mit einer Fehlermeldung informiert werden.

### Vorgehensweise

Fügen Sie im PAI den folgenden Codeblock vor der Zeile `LOOP AT mytable INTO textstruct` ein und legen Sie das Textelement mit einer entsprechenden Fehlermeldung an:

```
CALL METHOD EDITOR->GET_FIRST_VISIBLE_LINE
  IMPORTING
    LINE = INDEX
  EXCEPTIONS
    ERROR_CNTL_CALL_METHOD = 1.
IF FROM_IDX = INDEX.
  MESSAGE i208(00) WITH TEXT-003.
  EXIT.
ENDIF.
```

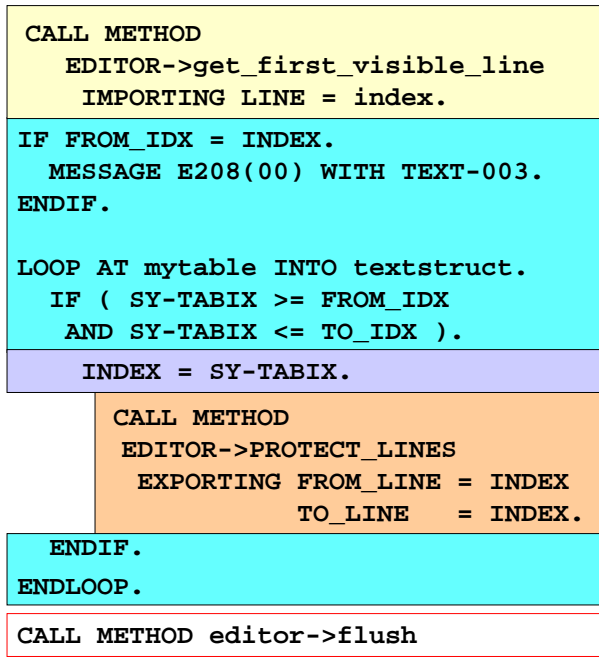
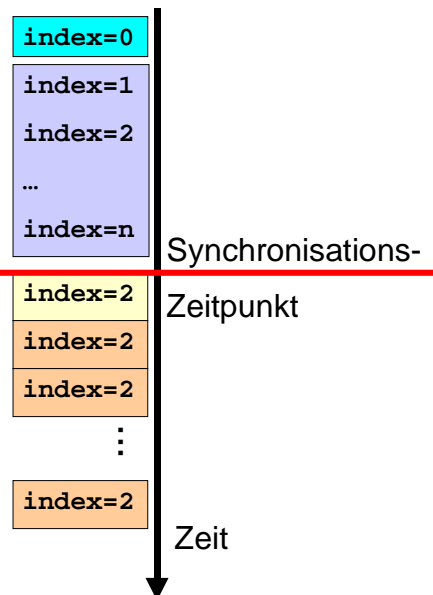
### Überprüfen Sie Ihre Arbeit

Starten Sie Ihr Programm, importieren Sie die interne Tabelle und markieren Sie einen beliebigen Bereich im Textfenster. Statt daß das System eine Fehlermeldung ausgibt, wird immer nur die erste sichtbare Zeile geschützt.

### Diskussion

Der Grund für das Fehlverhalten ist die mehrfache Verwendung der Variablen `INDEX` und die zeitliche Entkopplung der Methoden (aus Platzgründen wurde auf den Parameter `ENABLE_EDITING_PROTECTED_TEXT` der Methode `PROTECT_LINES` in der grafischen Darstellung verzichtet):

## Übung 2: Wertübergabe zwischen Methoden

ABAP-ProgrammLaufzeitverhalten**Abfrage des importierten Wertes**

Die ausbleibende Fehlermeldung hat folgenden Grund: In der `IF`-Abfrage hat `INDEX` noch seinen Initialwert 0, weil die Methode `get_first_visible_line` erst zum Flushzeitpunkt ausgeführt wird. Die Fehlermeldung bleibt deshalb aus. Bei einem weiteren Versuch, Zeilen zu schützen, hat `INDEX` immer gerade den Wert vom letzten Durchlauf des `CASE`-Zweigs. Dieser Effekt wurde schon in der letzten Übung beschrieben.

**Parameterübergabe innerhalb der Automation Queue**

Aus folgendem Grund werden die falschen Zeilen geschützt: Innerhalb der Schleife werden je nach Belegung der Variablen `FROM_IDX` und `TO_IDX` ein oder mehrere Aufrufe der Methode `PROTECT_LINES` in die Automation Queue gestellt. Die Aktualparameter werden dabei als Referenz gespeichert, weil die Methode `get_first_visible_line` den gleichen Parameter `INDEX` vorher importiert hat. Während der Laufzeit wird `INDEX` dann an die Aktualparameter der Methode `PROTECT_LINES` weitergegeben.

Die Automation Queue übernimmt die Werte von Referenzparametern erst zum Flush-Zeitpunkt. Deshalb hat die Zuweisung `INDEX = SY-TABIX` praktisch keine Auswirkung. Die Aktualparameter von der Methode `PROTECT_LINES` werden zudem überschrieben, nachdem `get_first_visible_line` den Wert von `LINE` zurückliefert.

---

**Übung 2: Wertübergabe zwischen Methoden**

Generell vermindert die Eigenschaft der Automation Queue, Referenzparameter weiterreichen zu können, die Anzahl der Flush-Aufrufe. Es wird empfohlen diese Technik soweit es geht auszuschöpfen, um die Performance zu erhöhen.

**Abhilfe**

Wenn Sie den Rückgabewert einer Methode nicht innerhalb der Automation Queue weitergeben möchten, können Sie:

- eine weitere Variable einführen, oder
- einen weiteren Flush einbauen.

Die Methodenaufrufe `get_selection_pos` und `get_first_visible_line` sind unabhängig voneinander. Stellen Sie deshalb beide in die Automation Queue, indem Sie:

den Flush-Aufruf, der auf die Methode `GET_SELECTION_POS` folgt, vor den `IF`-Block zur Ausgabe der Nachricht verschieben.

Danach werden wieder die markierten Zeilen geschützt und im Falle der ersten sichtbaren Zeile die angelegte Fehlermeldung in einem Popup ausgegeben.

## Übung 3: Control Methoden in Unterprogrammen

# Übung 3: Control Methoden in Unterprogrammen

## Verwendung

Die Methoden eines Control lassen sich auch in einem Unterprogramm beziehungsweise in einer Methode aufrufen. So kann man eine Funktion isolieren und Programme übersichtlicher machen.

In dieser Übung fassen Sie die Methodenaufrufe, die den Zustand des Control abfragen, in einem Unterprogramm zusammen.

## Vorgehensweise

1. Rufen Sie im PAI das Unterprogramm `get_lines` im `CASE`-Zweig auf, in dem Sie den Funktionscode `PROTECT` abfragen. Übergeben Sie dabei die Parameter `from_idx`, `to_idx` und `index` per Referenz:

```
WHEN 'PROTECT'.
  DATA: FROM_IDX TYPE I,
         TO_IDX TYPE I,
         INDEX TYPE I.
  PERFORM GET_LINES CHANGING FROM_IDX TO_IDX INDEX.
```

2. Legen Sie das Unterprogramm an und verschieben Sie die beiden Methodenaufrufe `GET_SELECTION_POS` und `GET_FIRST_VISIBLE_LINE` dorthin.



Achten Sie darauf, daß die formalen Parameter von `get_lines` mit den Aktualparametern der hineinkopierten Methoden übereinstimmen.

## Überprüfen Sie Ihre Arbeit

Das Verhalten des Programms ändert sich nicht. Dafür ist Ihr Coding etwas übersichtlicher geworden.

## Diskussion

Die Methoden `GET_SELECTION_POS` und `GET_FIRST_VISIBLE_LINE` importieren Werte in die Aktualparameter `from_idx`, `to_idx` und `index`. Das Control Framework speichert also beim Puffern der Methoden in der Automation Queue Referenzen auf diese Parameter ab.

## Referenzparameter des Unterprogramms

Die Zuordnung der Werte zu den Aktualparametern erfolgt zum Flush-Zeitpunkt, also in unserem Beispiel außerhalb des Unterprogramms. Etwas verwunderlich, denn die Aktualparameter der Methoden müssen ja zum Flushzeitpunkt gültig sein. Die Referenz eines Parameters des Unterprogramms ist aber nach Beendigung des Programms nicht mehr gültig; trotzdem gibt es keinen Laufzeitfehler.

Der Grund ist, daß die Referenzparameter von Unterprogrammen (bei Verwendung von `USING` und `CHANGING`) nach Außen an die Automation Queue durchgereicht werden. Die Referenzen sind also noch beim Flush der Automation Queue bekannt.

### Übung 3: Control Methoden in Unterprogrammen



Die Verwendung von globalen Variablen als Aktualparameter der Control-Methoden ist auch eine lauffähige Lösung. Das sollten Sie jedoch möglichst vermeiden, da Programme mit globalen Variablen unübersichtlicher sind.

#### Werteparameter von Unterprogrammen

Die Aktualparameter `from_idx`, `to_idx` und `index` werden nach dem Unterprogramm im PAI weiter verwendet. Deshalb ist es in unserem Beispiel nicht denkbar, die Parameter des Unterprogramms per call-by-value zu übergeben – die Rückgabewerte der Methoden `GET_SELECTION_POS` und `GET_FIRST_VISIBLE_LINE` gingen verloren.

Trotzdem wollen wir testweise einen der Parameter als Wert übergeben. Ändern Sie dazu die Definition der Form folgendermaßen:

```
FORM GET_LINES CHANGING VALUE(FROM_IDX) TO_IDX INDEX.
```

Starten Sie das Programm und versuchen Sie eine Zeile zu schützen. Das System bricht das Programm mit einem Laufzeitfehler ab. Das liegt daran, daß die Methode `GET_SELECTION_POS` die lokale Variable `FROM_IDX` in der Form verwendet, die beim Flush - also außerhalb der Form - nicht mehr gültig ist.



Dieses Problem tritt nur dann auf, wenn die Aktualparameter der Control-Methoden in der Automation Queue als Referenz gespeichert werden.

Die Konsequenz: Verwenden Sie in einem Unterprogramm lokale Variablen, die in der Automation Queue als Referenz gehalten werden, so muß innerhalb des Unterprogramms ein Flush-Aufruf stehen.

## Übung 4: Mehrere Controls verwenden

# Übung 4: Mehrere Controls verwenden

## Verwendung

In einem Programm können mehrere Controls gleicher oder verschiedener Art erzeugt werden. In dieser Übung legen Sie ein zweites Editorfenster an, das als Zwischenpuffer für kleinere Texte dienen soll.



Eine detailliertere Beschreibung zur Erzeugung eines Control finden Sie in [Übung 2: Ein Control und dessen Container erzeugen \[Seite 17\]](#) der ersten Lektion.

## Vorgehensweise

1. Legen Sie im Rahmenprogramm eine neue Referenzvariable `scratch` auf die Klasse des Textedit Controls und eine Referenzvariable `custom_container2` für das Custom Container Control an:

```
DATA: scratch TYPE REF TO cl_gui_textedit,
      custom_container2 TYPE REF TO cl_gui_custom_container.
```

2. Legen Sie auf dem Dynpro einen neuen Container `MYCONTAINER2` für ein Custom Control an.
3. Erzeugen Sie das Control (inklusive des Custom Container Control) zum PBO Zeitpunkt und blenden Sie die Statuszeile aus. Kopieren Sie dafür den folgenden Code in den `IF`-Block `IF EDITOR IS INITIAL`:

```
CREATE OBJECT custom_container2
  EXPORTING
    CONTAINER_NAME = 'MYCONTAINER2'
  EXCEPTIONS
    CNTL_ERROR = 1
    CNTL_SYSTEM_ERROR = 2
    CREATE_ERROR = 3
    LIFETIME_ERROR = 4
    LIFETIME_DYNPRO_DYNPRO_LINK = 5.

CREATE OBJECT SCRATCH
  EXPORTING
    parent = custom_container2
    WORDWRAP_MODE = CL_GUI_TEXTEDIT=>WORDWRAP_AT_WINDOWBORDER
    WORDWRAP_TO_LINEBREAK_MODE = CL_GUI_TEXTEDIT=>TRUE.
CALL METHOD SCRATCH->SET_STATUSBAR_MODE
  exporting
    STATUSBAR_MODE = CL_GUI_TEXTEDIT=>FALSE.
```

4. Aktivieren Sie die neuen Objekte und starten Sie Ihr Programm.

## Übung 4: Mehrere Controls verwenden

## Überprüfen Sie Ihre Arbeit

In dem zweiten Container wird ein Texteditor ohne Statuszeile angezeigt. Über die Funktionen **Kopieren** des ersten und **Einfügen** des zweiten Editors können Sie Text in das zweite Fenster kopieren.

## Diskussion

Im PBO werden mehrere Controls erzeugt und weitere Control-Methoden aufgerufen. Für die Übertragung der Methoden zum Frontend wird dabei lediglich eine Synchronisation am Ende des PBO benötigt. Das bedeutet, daß die Methoden von zwei verschiedenen Instanzen die gleiche Automation Queue benutzen.



Grundsätzlich gibt es pro internen Modus eine eigene Automation Queue. In dieser puffert das System also Methodenaufrufe aller Control-Instanzen, die im gleichen internen Modus erzeugt wurden.

Theoretisch können Sie beliebig viele Methodenaufrufe in der Queue puffern. Das System schränkt allerdings die Laufzeit für RFC-Aufrufe ein. Wird diese Zeit überschritten, müssen Sie zusätzliche Synchronisierungspunkte einführen, weil sonst das Backend die Verbindung abbricht. Dies hängt aber im allgemeinen nicht mit der Anzahl der RFC-Aufrufe zusammen, sondern mit den Datenmengen, die bei einem solchen Aufruf übertragen werden müssen (zum Beispiel lange Texte).

## Strategie

Insbesondere bei der Verwendung von mehreren Controls gibt es eine Strategie, um die Anzahl der Flush-Aufrufe zu reduzieren. Häufig muß man Eigenschaften des Controls mit GET-Methoden abfragen bevor man mit SET-Methoden das weitere Verhalten des Controls festlegt. Da man nach einer GET-Methode die aktuellen Werte im ABAP-Programm benötigt, ist ein Flush unvermeidbar (siehe auch [Übung 1: Importierte Werte verwenden \[Seite 42\]](#)). Bei mehreren voneinander unabhängigen GET-Aufrufen gehen Sie am besten folgendermaßen vor:

1. Bündeln Sie alle GET-Aufrufe in Ihrem Coding.
2. Rufen Sie die Methode FLUSH auf, um die Methoden zum Frontend zu übertragen und die Werte der GET-Aufrufe zu importieren.
3. Fragen Sie diese Werte ab und rufen Sie in einem zweitem Block alle SET-Methoden auf.

Diese Gruppierung der Aufrufe ermöglicht so eine Optimierung der Performance.

Zusammenfassung

## Zusammenfassung

In dieser Lektion haben Sie gelernt, daß die Control-Methoden in einer Automation Queue gepuffert werden. Grundsätzlich hat die Queue folgende Eigenschaften:

- Das Control Framework überträgt die Queue zum Frontend, wenn Sie die Methode `CL_GUI_CFW=>FLUSH` aufrufen.
- Es existiert eine Automation Queue pro internen Modus. Das bedeutet, daß Sie Methoden verschiedener Control-Instanzen in einer Queue puffern können.
- Das Control Framework speichert die Aktualparameter als Referenzen oder Werte in der Automation Queue.
- Aktualparameter werden genau dann als Referenzen in der Queue gespeichert, wenn die Methode Rückgabewerte hat oder der Aktualparameter in einem vorhergehenden Aufruf der Queue schon einmal zurückgeliefert wurde. In allen anderen Fällen speichert das Framework die Aktualparameter als Wert in der Queue.

Die beiden letzten Punkte haben Konsequenzen für die Wahl des richtigen Flushzeitpunktes bzw. der weiteren Verwendung von Variablen, die als Aktualparameter genutzt werden:

- Wenn Sie Variablen für die Rückgabewerte von Control-Methoden verwendet haben, sollten Sie diese nur noch als Aktualparameter für die folgenden Control-Methoden verwenden.
- Benutzen Sie keine Systemvariablen als Aktualparameter.
- Wenn Sie in Unterprogrammen Werte vom Control abfragen, müssen bei der Verwendung von lokalen Variablen unbedingt einen Flush ausführen. Die folgende Tabelle faßt dies noch einmal zusammen:

### Zwingend nötige Synchronisierung vor dem Beenden von Unterprogrammen

<b>Art der Speicherung in der Automation Queue:</b>	<b>Lokale Variable im Unterprogramm oder Übergabe per <code>value()</code></b>	<b>Globale Variable im Unterprogramm Oder Übergabe per Referenz</b>
<b>Als Wert</b>	Flush nicht zwingend nötig.	Flush nicht zwingend nötig.
<b>Als Referenz</b>	Flush vor dem Verlassen des Unterprogramms bzw. vor jedem <b>EXIT</b> zwingend notwendig.	Flush nicht zwingend nötig.

## Lektion 4: Testen und Fehlerbehandlung

### Aufgabenstellung

Lassen Sie sich die Methoden anzeigen, die in der Automation Queue übertragen werden.

### Überblick: Lerninhalte

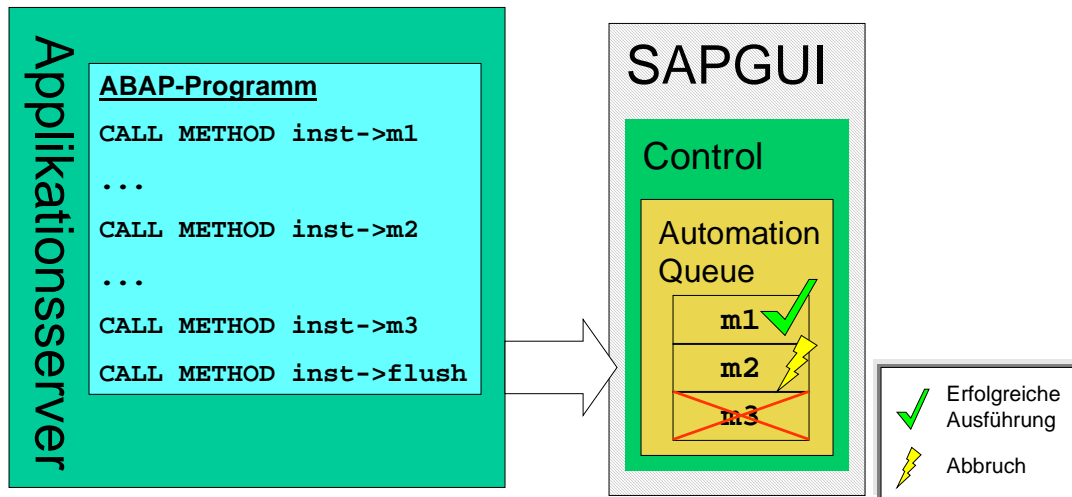
In dieser Lektion lernen Sie, was beim Debuggen von Programmen, die Controls verwenden, zu beachten ist und welche Möglichkeiten Ihnen die ABAP Workbench bei der Fehlersuche bietet.

**Siehe auch:** [Services rund um die Automation Queue \[Extern\]](#) der Control Framework Dokumentation

## Einführung

## Einführung

## R/3-Architektur



## Erläuterung

Nachdem das Control Framework die Methoden per RFC zum SAPGUI übertragen hat, werden sie nacheinander ausgeführt. Kommt es bei einer Methode zu einem Laufzeitfehler am Frontend, bricht die Verarbeitung der Methoden ab. Das heißt, daß sowohl die Methode, bei der der Fehler aufgetreten ist, als auch nachfolgende Methoden nicht weiter ausgeführt werden.

Aus der Sicht des Backends tritt ein Fehler erst beim Aufruf der Methode **FLUSH** auf, da die Control-Methoden vorher nur gepuffert aber nicht ausgeführt werden.



Der Abschnitt [Fehlerbehandlung bei der Synchronisation \[Extern\]](#) der Control Framework Dokumentation geht genauer auf die Fehlerbehandlung im Zusammenhang mit **impliziten** Flush-Aufrufen ein.

## Folgerungen

Wenn bei einem Flush-Aufruf eine Exception ausgelöst wird, ist noch nicht geklärt, bei welcher Methode innerhalb der Automation Queue ein Fehler aufgetreten ist. Aus diesem Grund können Sie diese Laufzeitfehler nicht dynamisch abfangen und darauf reagieren. Zum Flush-Zeitpunkt ist lediglich klar, daß ein Fehler aufgetreten ist, aber nicht an welcher Stelle.



## Übung 1: Den Trace Modus aktivieren

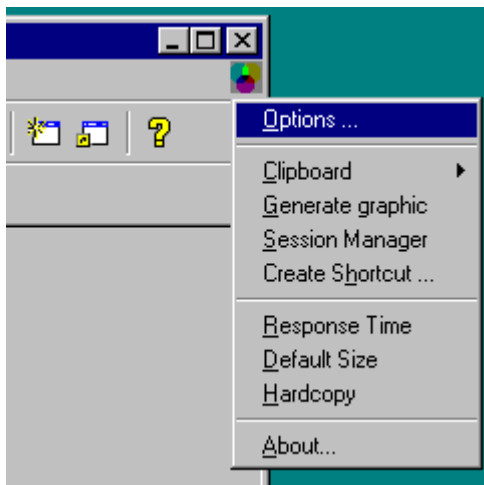
## Übung 1: Den Trace Modus aktivieren

### Verwendung

Um verfolgen zu können, welche Methodenaufrufe in die Automation Queue gestellt werden, können Sie den *Trace Modus* verwenden. Alle Automationaufrufe werden mit Parametern und Ergebniswerten in eine Datei geschrieben und in einem Fenster angezeigt. Dies ist zum Beispiel hilfreich, um zu erkennen, bei welcher aufgerufenen Methode in der Queue ein Fehler aufgetreten ist.

### Vorgehensweise

1. Wählen Sie Options im Konfigurationsmenü:



Das System öffnet ein Fenster mit einem Tabstrip für die allgemeinen Optionen.

2. Wählen Sie den Tab-Reiter *Trace*.
3. Um den Trace Modus einstellen zu können klicken Sie zunächst auf den Knopf *Ausschalten*.
4. Markieren Sie das Ankreuzfeld *Automation* und wählen Sie einen Anzeigeelevel über den Radiobutton *Trace Level*.



Bei Level 1 zeigt das System alle Automation-Aufrufe inklusive Parameter und Ergebniswerten an. Bei Level 2 werden zusätzlich zu Level 1 die Tabellen der Automationqueue ausgegeben.

5. Geben Sie einen Dateinamen für die Tracedatei an, in die die Informationen gespeichert werden sollen.



Die Datei wird auf Ihrem lokalen PC im Verzeichnis `C:\sappcadm` angelegt.

---

**Übung 1: Den Trace Modus aktivieren**

6. Um die Trace Informationen während des Programmablaufs verfolgen zu können, markieren Sie das Ankreuzfeld *Fenster*.

7. Klicken Sie auf den Knopf *Anschalten*.

Das System öffnet ein zusätzliches Fenster, in dem die Automation Aufrufe angezeigt werden.

8. Bestätigen Sie Ihre Angaben mit dem *OK* Knopf.

## Überprüfen Sie Ihr Ergebnis

Starten Sie Ihr Programm. Über das Trace-Fenster können Sie sehen, wie Ihre Methodenaufrufe über die Automation Queue an den Automation Controller weitergereicht werden. Wenn Sie einen Dateinamen angegeben haben, finden Sie in der erzeugten Datei den gleichen Text.



Die Übertragung der aufgelisteten Methoden wird durch eine horizontale Linie im Trace-Lauf gekennzeichnet, d.h. an diesen Stellen wird der Flush ausgelöst. Sie können den Trace-Lauf also auch zur Flushoptimierung einsetzen.

## Übung 2: Synchroner Verarbeitung der Methoden

# Übung 2: Synchroner Verarbeitung der Methoden

## Verwendung

Bei Control-Methoden hat die zeitliche Entkopplung von Methodenaufrufen und deren Ausführung zur Folge, daß bei einem Fehler am Frontend der Fehler erst beim Flush auftritt. Mit einer Synchronisation nach jeder Control-Methode kann man dagegen feststellen, bei welcher Methode die Verarbeitung abbricht.

Im Debugger haben Sie die Möglichkeit, eine automatische Synchronisation nach jeder Control-Methode zu aktivieren. Das System schiebt dann im Debugging-Modus nach jedem Control-Aufruf einen Flush-Aufruf ein. Bei einem Laufzeitfehler wird dann also der Aufruf, bei dem der Fehler aufgetreten ist, angezeigt.

Die Debugger-Einstellung zur synchronen Verarbeitung der Methoden ist aber nicht nur für das Aufspüren der Abbruchstelle interessant. Wenn Sie den Ablauf Ihres Programms im Debugger ohne synchrone Verarbeitung verfolgen, werden die Control-Methoden eben nur gepuffert und noch nicht ausgeführt. Der Sy-Subrc geht Ihnen somit verloren. Außerdem haben Variablen, die über die Methoden importiert werden, noch keine aktuellen Werte.



Es kann vorkommen, daß Ihr Programm bei synchroner Verarbeitung korrekt läuft, aber ansonsten nicht. In diesem Fall fehlt meist ein expliziter Flush-Aufruf.

## Vorgehensweise

1. Klicken Sie im ABAP Debugger auf die Drucktaste `Einstellungen`.
2. Im Rahmen für die Debugger-Einstellungen markieren Sie das Ankreuzfeld `Automation Controller: Aufträge immer synchron verarbeiten`.
3. Aktivieren Sie zusätzlich den Trace-Modus (siehe [Übung 1: Den Trace Modus aktivieren](#) [\[Seite 56\]](#)).

## Überprüfen Sie Ihre Arbeit

Im Trace-Lauf sehen Sie, daß nach jeder Control-Methode ein Flush ausgeführt wird.

## Zusammenfassung

Die ABAP Workbench bietet Ihnen folgende Möglichkeiten beim Debugging von Programmen, die Controls benutzen:

- Mit dem Trace Modus können Sie mitverfolgen, welche Control-Methoden mit der Automation Queue übergeben werden und ob sie erfolgreich ausgeführt werden. Die Synchronisierungs-Zeitpunkte sind durch eine horizontale Linie gekennzeichnet.
- Mit einem Flag im Debugger können Sie die Automation-Aufrufe synchron verarbeiten. Bei einem Laufzeitfehler wird dann die richtige Control-Methode als Abbruchstelle angegeben.

---

**Weitere Informationsquellen**

## Weitere Informationsquellen

Übergreifende Dokumentation:

- [SAP Control Framework \[Extern\]](#)
- [SAP Container \[Extern\]](#)

Klassendokumentationen für Basis-Controls:

- [SAP HTML Viewer Control \[Extern\]](#)
- [SAP Picture Control \[Extern\]](#)
- [SAP TextEdit Control \[Extern\]](#)
- [SAP Tree Control \[Extern\]](#)