



ABAP für Fortgeschrittene

**Teil 5: Oberflächen mit ALV Grid View, Dynam.
Programmierung, Fallstudie**

Copyright

- *Das vorliegende Skriptum baut zu großen Teilen auf den Publikationen zum TAW11- und TAW12-Kurs – das Copyright dieser Teile liegt bei der SAP AG.*
- *Die in diesem Kurs verwendeten Abbildungen wurden – falls nicht anders gekennzeichnet – in Anlehnung zum TAW11- und TAW12-Kurs erstellt. Das Copyright dieser Teile liegt bei der SAP AG.*
- *Für alle Screenshots im Skriptum, auch wenn diese nur verkürzt oder auszugsweise gezeigt werden, gilt der Hinweis: Copyright SAP AG*
- *Die Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die schriftliche Genehmigung von Prof. Dr. Heimo H. Adelsberger, Dipl.-Wirt.-Inf. Pouyan Khatami und Dipl.-Wirt.-Inf. Taymaz Khatami nicht gestattet..*

Inhaltsverzeichnis

COPYRIGHT	2
INHALTSVERZEICHNIS	3
ABBILDUNGSVERZEICHNIS	5
8 ALV GRID CONTROL	7
8.1 ARCHITEKTUR	7
8.2 PRAXIS: ÜBUNG ZUR AUSGABE VON DATEN PER ALV GRID CONTROL	15
8.3 EIGENE FELDKATALOGE	17
8.3.1 Angaben für nicht-Strukturfelder im Feldkatalog	19
8.3.2 Texte im Feldkatalog	19
8.3.3 Währungs- und Mengenangaben im Feldkatalog	20
8.3.4 Weitere Angaben im Feldkatalog	21
8.4 PRAXIS: ÜBUNG ZUM FELDKATALOG	22
8.5 EREIGNISSE DES ALV GRID CONTROL	25
8.6 PRAXIS: ÜBUNG ZUR EREIGNISBEHANDLUNG	27
8.7 ERWEITERN DER TOOLBAR	29
8.8 PRAXIS: ÜBUNG ZUM ERWEITERN DER TOOLBAR	30
8.9 WEITERE EREIGNISSE UND METHODEN DES ALV GRID CONTROL	31
8.10 PRAXIS: ÜBUNG ZUR FORMATIERUNG DER DRUCKLISTE	34
8.11 KONTROLLFRAGEN	35
8.12 ANTWORTEN	37
9 ABAP UND UNICODE	38
10 DYNAMISCHE PROGRAMMIERUNG	41
10.1 FELDSYMBOLS	41
10.2 DATENREFERENZEN	45
10.3 PRAXIS: ÜBUNG ZUR TYPISIERUNG VON DATENREFERENZEN	49
10.4 PRAXIS: ÜBUNG ZUR AUSGABE VON TABELLENINHALTEN	51
10.5 RUN TIME TYPE SERVICES (RTTS)	53
10.5.1 Run Time Type Identification (RTTI)	53
10.5.2 Praxis: Übung zur Ausgabe der Komponenten einer Struktur	54
10.5.3 Praxis: Übung zur Ausgabe von Informationen über Klassen	55
10.5.4 Run Time Type Creation (RTTC)	57
10.6 PRAXIS: ÜBUNG ZUR TYPERZEUGUNG ZUR LAUFZEIT	58
10.7 KONTROLLFRAGEN	61
10.8 ANTWORTEN	62
11 SYSTEM- UND UNTERNEHMENSÜBERGREIFENDE GESCHÄFTSPROZESSE	63
11.1 APPLICATION LINK ENABLING	63
11.2 BUSINESS OBJECTS UND BAPIS	64
11.3 PRAXIS: ÜBUNG ZU BAPIS	66
11.4 ENTERPRISE SOA UND WEB SERVICES	68

12	DER CODE INSPECTOR	71
12.1	PRAXIS: ÜBUNG ZUM CODE INSPECTOR.....	73
12.2	FAZIT: CODE INSPECTOR	75
12.3	KAPITELABSCHLUSS	76
13	PRAXIS: FALLSTUDIE UNIVERSITÄT	77
13.1	SZENARIO	77
13.2	AUFGABE.....	77
13.3	KAPITELABSCHLUSS	79

Abbildungsverzeichnis

Abbildung 1: Architektur des Control Frameworks	7
Abbildung 2: Drei Stufen zum EnjoySAP-Control	8
Abbildung 3: Einbinden eines ALV Grid Controls in ein Dynpro	8
Abbildung 4: Darstellung eines Custom Control-Bereichs im Screen painter	9
Abbildung 5: Sequenzdiagramm: Ablauf der Container Control-Instanziierung	9
Abbildung 6: Beispiel zur Containerobjekterzeugung	10
Abbildung 7: Sequenzdiagramm mit Prüfung auf Initialwert	11
Abbildung 8: Erzeugen des ALV Grid Control-Objekts	11
Abbildung 9: Elemente auf GUI- und ABAP-Seite	12
Abbildung 10: Herkunft der Daten eines ALV Grid Control	13
Abbildung 11: Aufruf der Methode set_table_for_first_display	14
Abbildung 12: Aktualisieren der Darstellung	15
Abbildung 13: Erstellen des Containerobjekts: SAP-System-Screenshot	16
Abbildung 14: Muster zum Aufruf einer Methode: SAP-System-Screenshot	16
Abbildung 15: Ihr erstes ALV Grid Control: SAP-System-Screenshot	17
Abbildung 16: Zusammenhang zwischen Datentabelle und Feldkatalog	18
Abbildung 17: Erforderliche Parameter zur Feldkatalog-Erstellung	18
Abbildung 18: Beispieleinträge in einem Feldkatalog	19
Abbildung 19: Formatierung von Währungsfeldern	21
Abbildung 20: Oberfläche des Programms: SAP-System-Screenshot	24
Abbildung 21: Verringerte Spaltenbreiten: SAP-System-Screenshot	25
Abbildung 22: Behandlung von Ereignissen des ALV Grid Control	26
Abbildung 23: Klick-Ereignisse und Mauszeiger	27
Abbildung 24: Dynpro 300 im Screen Painter: SAP-System-Screenshot	27
Abbildung 25: Doppelklick auf einen Eintrag: SAP-System-Screenshot	28
Abbildung 26: Anzeige der Details des gewählten Kunden: SAP-System-Screenshot	29
Abbildung 27: Toolbar eines ALV Grid Controls: SAP-System-Screenshot	29
Abbildung 28: Neuer Button in der Toolbar: SAP-System-Screenshot	30
Abbildung 29: Ausgabe der Bonuspunkte: SAP-System-Screenshot	31
Abbildung 30: Aufruf der Listausgabe: SAP-System-Screenshot	32
Abbildung 31: Listausgabe des ALV Grid Control: SAP-System-Screenshot	32
Abbildung 32: Ereignisse der Listausgabe	33
Abbildung 33: Ausgabe von Reisebüroname und Datum: SAP-System-Screenshot	34
Abbildung 34: Sortierung definieren: SAP-System-Screenshot	35
Abbildung 35: Gruppierete Einträge: SAP-System-Screenshot	35
Abbildung 36: Aktivieren der Unicodeprüfungen: SAP-System-Screenshot	38
Abbildung 37: Byte- und Character-Mode	38
Abbildung 38: Fragmente von Strukturen	39
Abbildung 39: Zugriff mit Offset und Länge	40
Abbildung 40: Anwendungsbeispiel für Feldsymbole	42
Abbildung 41: Code mit Syntaxfehler	42
Abbildung 42: Fehlermeldung bei der Syntaxprüfung: SAP-System-Screenshot	43
Abbildung 43: Code, der zu einem Laufzeitfehler führt	43
Abbildung 44: Laufzeitfehler: SAP-System-Screenshot	43
Abbildung 45: Der CASTING-Zusatz	44
Abbildung 46: CASTING bei generischer Typisierung	44
Abbildung 47: Dynamische Typangabe beim Casting	45
Abbildung 48: Verwendung von GET REFERENCE und Dereferenzierungs-Operator	46
Abbildung 49: Dereferenzierung einer generischen Datenreferenz	47
Abbildung 50: Zugriff auf Strukturkomponenten per Datenreferenz	47
Abbildung 51: Zuweisung von Datenreferenzen	48
Abbildung 52: Erster Quellcode: SAP-System-Screenshot	49

Abbildung 53: F1-Hilfe: SAP-System-Screenshot.....	50
Abbildung 54: Zuweisung zur generischen Datenreferenz und Ausgabe: SAP-System-Screenshot	50
Abbildung 55: F1-Hilfe auf der zweiten Ausgabe: SAP-System-Screenshot	50
Abbildung 56: Direktes holen der Referenz: SAP-System-Screenshot.....	50
Abbildung 57: F1-Hilfe auf der dritten Ausgabe: SAP-System-Screenshot	51
Abbildung 58: Ausgaben des Programms: SAP-System-Screenshot.....	52
Abbildung 59: Beispiel für eine Lösung der Aufgabe: SAP-System-Screenshot	53
Abbildung 60: Hierarchie der RTTI-Beschreibungsklassen	54
Abbildung 61: Ausgabe des Programms: SAP-System-Screenshot.....	55
Abbildung 62: Ausgabe des Klassennamens: SAP-System-Screenshot	56
Abbildung 63: Ausgabe des Oberklassennamens: SAP-System-Screenshot	56
Abbildung 64: Ausgabe eines weiteren Oberklassennamens: SAP-System-Screenshot.....	57
Abbildung 65: Methoden zur Erzeugung von Typobjekten	57
Abbildung 66: CREATE-Methode im Navigationsbaum: SAP-System-Screenshot	58
Abbildung 67: Variablenanzeige im Debugger: SAP-System-Screenshot.....	59
Abbildung 68: Anzeige der Tabelle im Debugger: SAP-System-Screenshot	60
Abbildung 69: Anwendungsszenario für ALE	63
Abbildung 70: RFC-Aufruf eines Funktionsbausteins	66
Abbildung 71: Suche im BAPI-Explorer: SAP-System-Screenshot	67
Abbildung 72: Ergebnis der Suche: SAP-System-Screenshot	67
Abbildung 73: Navigationsbaum des BAPI-Explorers: SAP-System-Screenhshot	67
Abbildung 74: Nutzungsszenario von Web Services	69
Abbildung 75: Aufruf des Code Inspectors: SAP-System-Screenshot.....	71
Abbildung 76: Ergebnisse des Code Inspectors: SAP-System-Screenshot	71
Abbildung 77: Einstiegsbild des Code Inspectors: SAP-System-Screenshot.....	72
Abbildung 78: Details zu einer Warnung: SAP-System-Screenshot.....	73
Abbildung 79: Konfiguration der Suche: SAP-System-Screenshot	74
Abbildung 80: Anlegen der Inspektion: SAP-System-Screenshot	75

8 ALV Grid Control

In diesem Kapitel lernen Sie mit dem **ALV Grid Control** ein weiteres Element zur Gestaltung von Dynpros kennen. ALV ist ein Akronym für **SAP List Viewer**. Es handelt sich dabei nicht um ein simples GUI-Element wie etwa ein Textelement oder ein Eingabefeld, sondern um ein leistungsfähiges Werkzeug zur Darstellung von Tabellarischen Inhalten. Es bietet bereits vordefinierte Funktionen für typische Aufgaben in Zusammenhang mit Tabellen, kann aber auch um eigene Funktionen erweitert werden und ist daher für vielfältige Einsatzgebiete verwendbar.

8.1 Architektur

Seit dem Release 4.6A wird das Control Framework (CFW) im SAP-System eingesetzt um die Kommunikation mit allen Controls zu realisieren. Es können mit dieser Architektur nicht nur die einfachen GUI-Elemente, die Sie aus dem Screen Painter kennen, verwendet werden, sondern auch komplexe Elemente, die eigenständige Softwarekomponenten sind. Diese Elemente werden dann in bestimmte Bereiche auf einem Dynpro geladen. Zu diesen EnjoySAP-Controls genannten Elementen zählen etwa Picture Control, Textedit Control oder das in diesem Kapitel betrachtete ALV Grid Control.

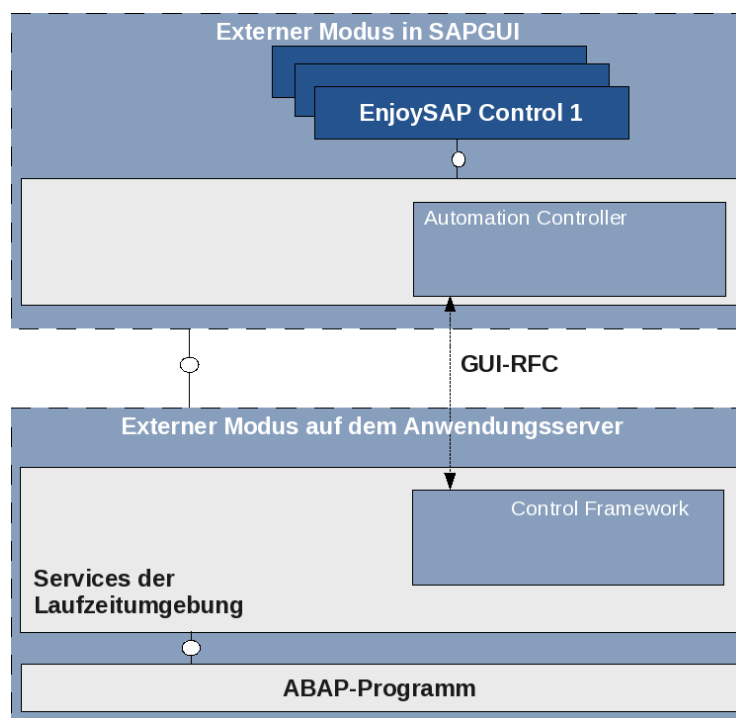


Abbildung 1: Architektur des Control Frameworks

Der Automation Controller ist Teil des Frontends und nimmt Verwaltungsaufgaben für die eingebundenen Controls wahr. Über ihn läuft die gesamte Kommunikation zwischen den Controls auf dem Frontend und dem ABAP-Programm. Methodenaufrufe aus dem ABAP-Programm an ein Control laufen über das Control Framework und können von diesem in einer Queue gesammelt werden, um nicht für jeden Aufruf sofort mit dem Frontend kommunizieren zu müssen.

Um nun ein Control wie das ALV Grid Control in einem Dynpro zu verwenden, ist eine mehrstufige Konfiguration vorzunehmen. Auf dem Dynpro ist ein Custom-Control-Bereich zu

reservieren, in dem das Control später erscheinen soll. In diesen Bereich wird dann ein Container Control geladen. Diesem Container Control wird dann das Control (hier ALV Grid Control) zugeordnet. Jede Instanz eines Controls korrespondiert mit einer repräsentativen Instanz im ABAP-Programm.

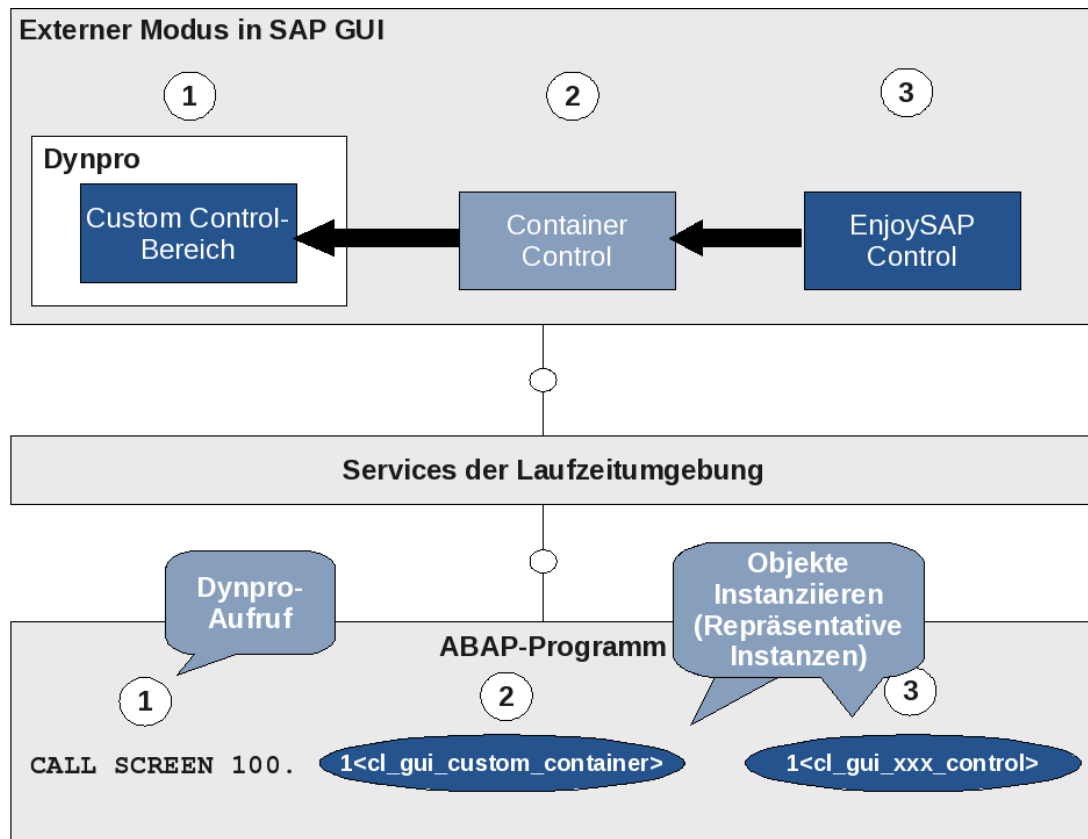


Abbildung 2: Drei Stufen zum EnjoySAP-Control

Die folgende Abbildung Veranschaulicht das soeben beschriebene noch einmal aus Sicht des Dynpros.

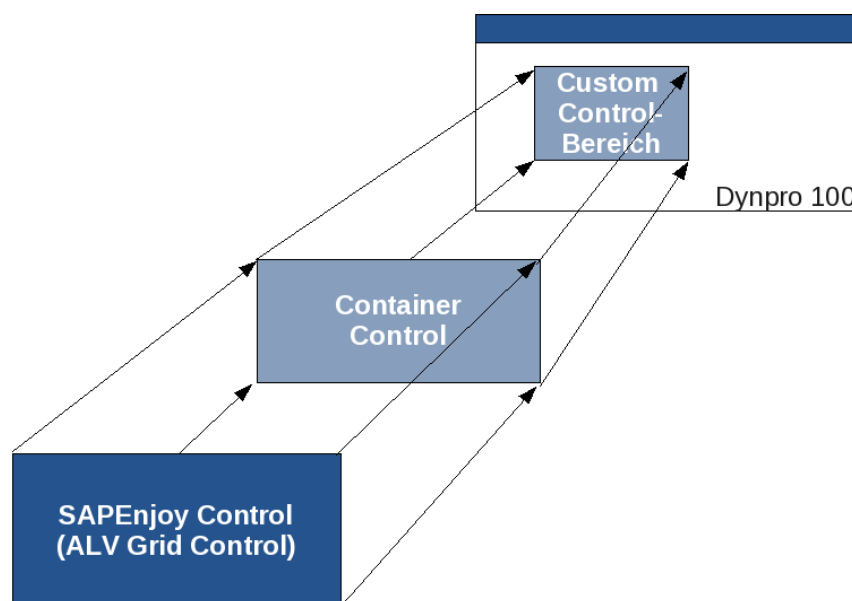


Abbildung 3: Einbinden eines ALV Grid Controls in ein Dynpro

Um einen Custom Control-Bereich zu definieren, wird im Screen Painter die Schaltfläche



verwendet und der Bereich durch Klicken und Ziehen markiert. Der Bereich ist dann als Rechteck mit einem diagonalen Kreuz sichtbar:



Abbildung 4: Darstellung eines Custom Control-Bereichs im Screen painter

Um im reservierten Bereich das benötigte Container Control zu erzeugen, wird ABAP-Code benötigt, in dem eine Instanz der entsprechenden Klasse erzeugt wird. Das Erzeugen der Instanz erfolgt im PBO-Block. Es ist empfehlenswert, den Code in einem separaten PBO-Modul unterzubringen, ihn also von anderen zu PBO ausgeführten Aufgaben zu trennen, um die Übersichtlichkeit des Programms zu gewährleisten. Der Ablauf des Programms wird im folgenden Sequenzdiagramm veranschaulicht.

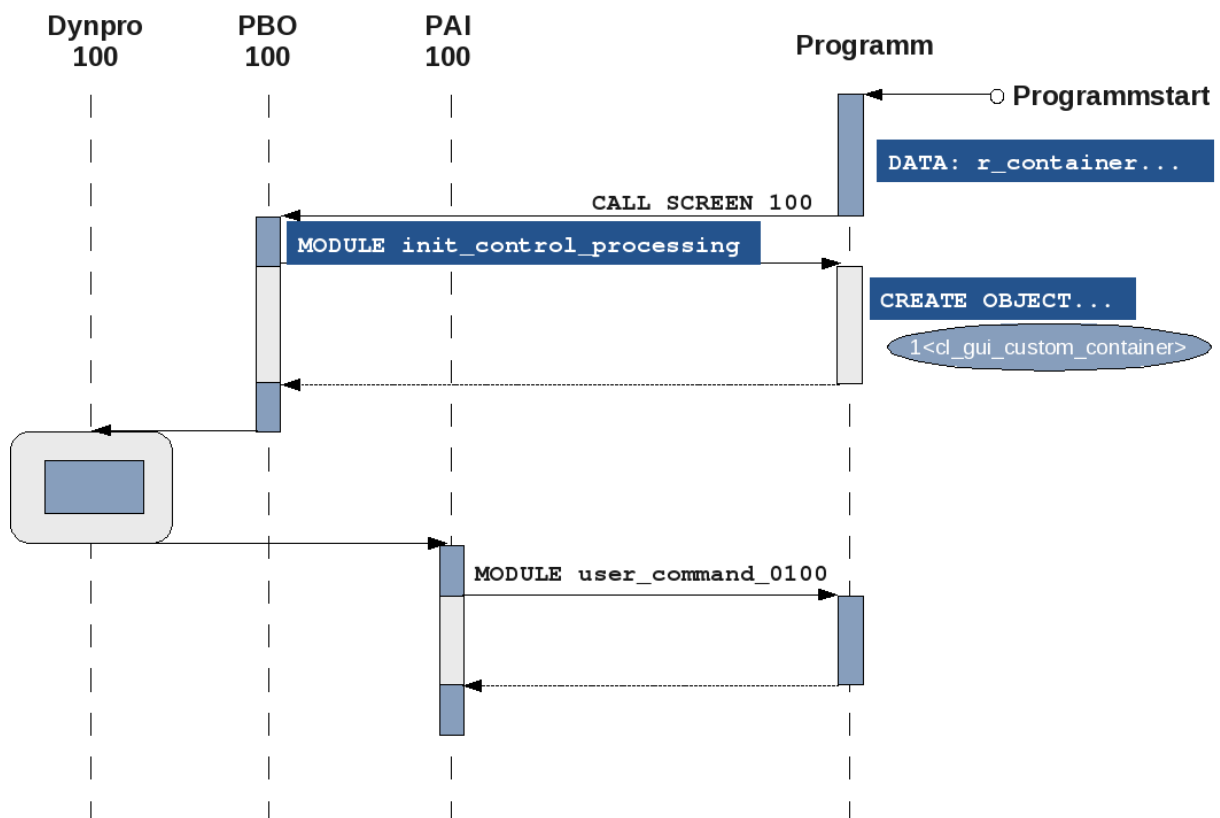


Abbildung 5: Sequenzdiagramm: Ablauf der Container Control-Instanziierung

Die Referenzvariable zur Erzeugung der Container Control-Instanz wird über die Klasse **cl_gui_custom_container** typisiert. Der Instanzkonstruktor dieser Klasse erhält einen Parameter **container_name**. Über diesen Parameter wird der Name des Custom Control-Bereichs auf der Dynpro-Oberfläche angegeben, in den das Container Control eingefügt werden soll. Die folgende Abbildung zeigt die Aufrufsyntax beispielhaft.

```
* Deklaration der Referenzvariablen
DATA: r_container
      TYPE REF TO cl_gui_custom_container.

* Im PBO des Dynpros
MODULE init_control_processing OUTPUT.
  IF r_container IS INITIAL.
    * Erzeugen des Containerobjekts und Anknüpfung an
    * den Bereich
    CREATE OBJECT r_container
      EXPORTING
        container_name = 'MY_CONTROL_AREA'
      EXCEPTIONS
        others = 1.
    IF sy-subrc NE 0.
      MESSAGE a...
    ENDIF.
    ...
  ENDIF.
```

Abbildung 6: Beispiel zur Containerobjekterzeugung

Sie sehen im hier dargestellten Code eine Prüfung, ob `r_container` initial ist. Häufig ist das Dynpro mit dem Container sein eigenes Folgedynpro. Durch die Prüfung soll verhindert werden, dass bei einem erneuten Auslösen von PBO durch eine entsprechende Aktion auf dem Dynpro ein neues Objekt erzeugt wird.

Die folgende Abbildung zeigt das Sequenzdiagramm mitsamt der Prüfung.

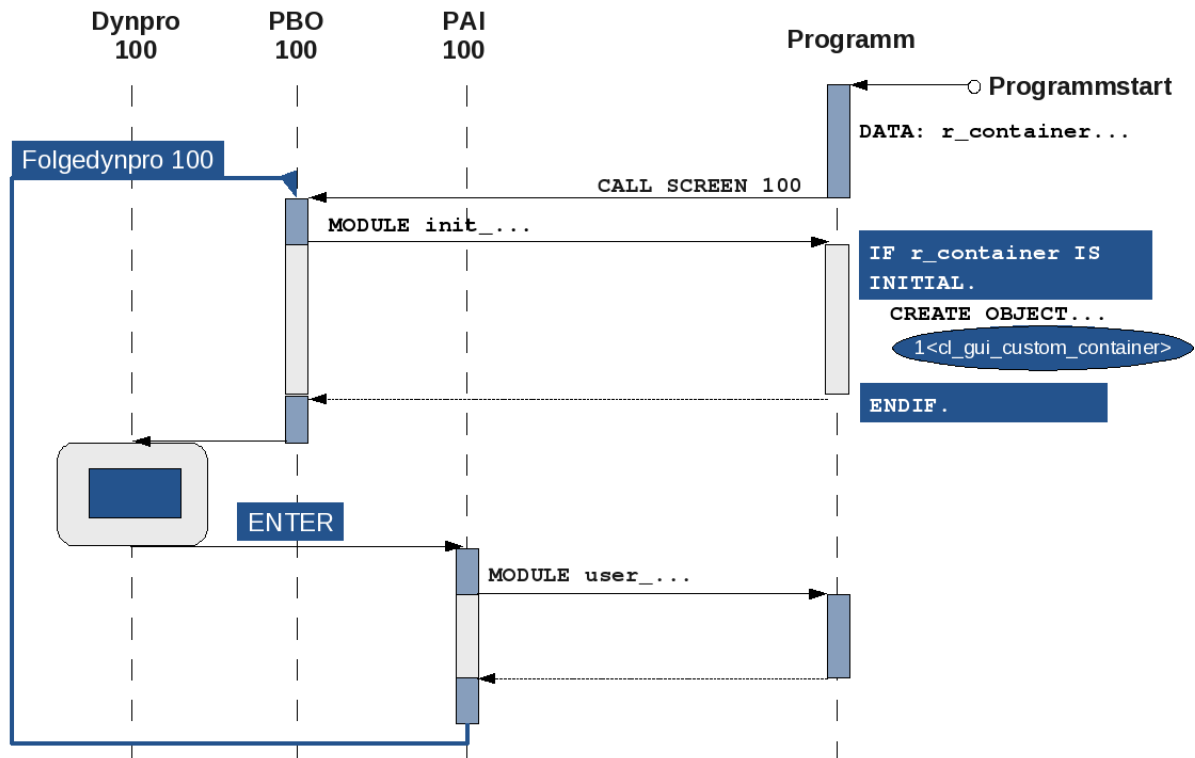


Abbildung 7: Sequenzdiagramm mit Prüfung auf Initialwert

Für das ALV Grid Control muss ebenfalls ein Objekt angelegt werden. Dieses wird über die Klasse **cl_gui_alv_grid** typisiert. Das Vorgehen ist dem beim Anlegen des Containerobjekts recht ähnlich. Wichtig ist, dass dieses Objekt erst nach dem Containerobjekt erzeugt wird, da der Konstruktor des ALV-Objekts eine Referenz auf das Containerobjekt benötigt. Die Syntax zur Erzeugung zeigt das folgende Codebeispiel.

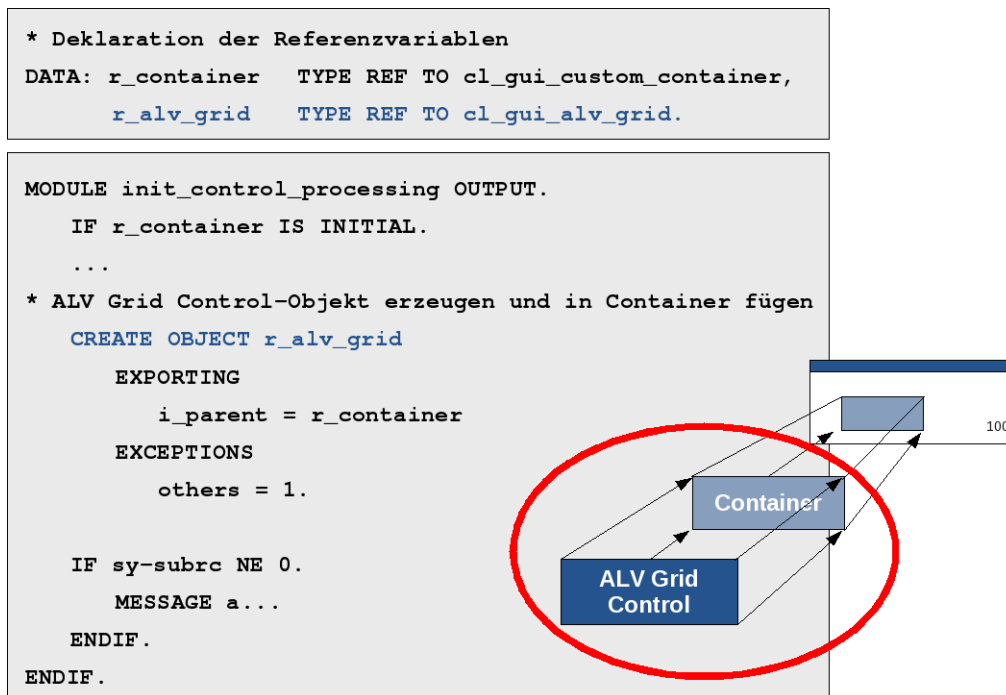


Abbildung 8: Erzeugen des ALV Grid Control-Objekts

Durch diese Schritte befindet sich nun auf dem von SAP GUI angezeigten Dynpro ein Custom Control-Bereich, in dem sich ein Container Control befindet, das wiederum das ALV

Grid Control enthält. Auf Seiten des ABAP-Programms existieren die beiden Objekte für den Container und für das ALV Grid Control.

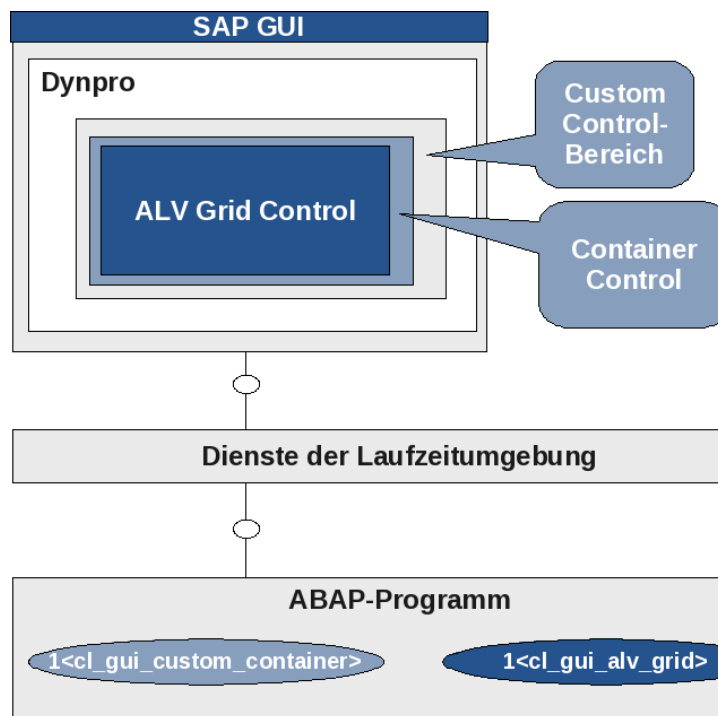


Abbildung 9: Elemente auf GUI- und ABAP-Seite

Diese repräsentativen Objekte (auch Proxy-Objekte genannt) kapseln den Kommunikationsprozess, so dass aus dem ABAP-Programm nur mit diesen Objekte gearbeitet werden muss. Die Objekte kommunizieren auch mit dem Spool-System, um bei Bedarf Listen drucken zu können. Alle dafür benötigte Funktionalität ist in der entsprechenden Klasse implementiert. Wenn Sie die Klassen im Class Builder anschauen, sehen Sie den immensen Umfang an Methoden und Attributen. Trotz alledem sind die Daten, die im ALV Grid Control dargestellt werden sollen, nicht im Objekt selbst hinterlegt. Diese werden stattdessen aus dem Objekt referenziert. Die zur Anzeige bereitgestellten Daten müssen daher während der Lebensdauer des ALV Grid vorgehalten werden. Die Änderungen, die an der Oberfläche an den Grid-Inhalten vollzogen werden, also etwa ein Umsortieren durch die entsprechenden Oberflächenfunktionen, werden hierbei stets an den Daten des Programms vollzogen. Es muss sich bei den übergebenen Daten also um schreibbare Daten handeln. Ungeeignet wären hier Daten aus dem zum Lesen geöffneten Shared Memory. Diese müssten erst in eine separate interne Tabelle kopiert werden. Die Anforderung an schreibbare Daten ist insbesondere unabhängig davon, ob tatsächlich Änderungen an den Daten im ALV Grid Control vorgesehen sind. Die Daten die im ALV Grid Control dargestellt werden sollen, müssen sich in einer internen Tabelle befinden.

Hinweis: Das Sortieren von internen Tabellen ist natürlich nicht nur mit ALV Grid Control-Oberflächen, sondern auch in ABAP-Quellcode möglich. Der Befehl hierfür lautet `SORT itab [BY spaltenname [DESCENDING|ASCENDING]]`. Mit dem Zusatz `AS TEXT` kann eine lexikographische Sortierung erzwungen werden, so dass „ä“ wie im deutschen üblich vor „b“ und nicht nach „z“ sortiert wird. Mit dem Zusatz `STABLE` bleibt die Reihenfolge der Datensätze, die sich im Sortierschlüssel nicht unterscheiden, erhalten. Weitere Informationen zu diesem Befehl erhalten Sie in der Schlüsselwortedokumentation.

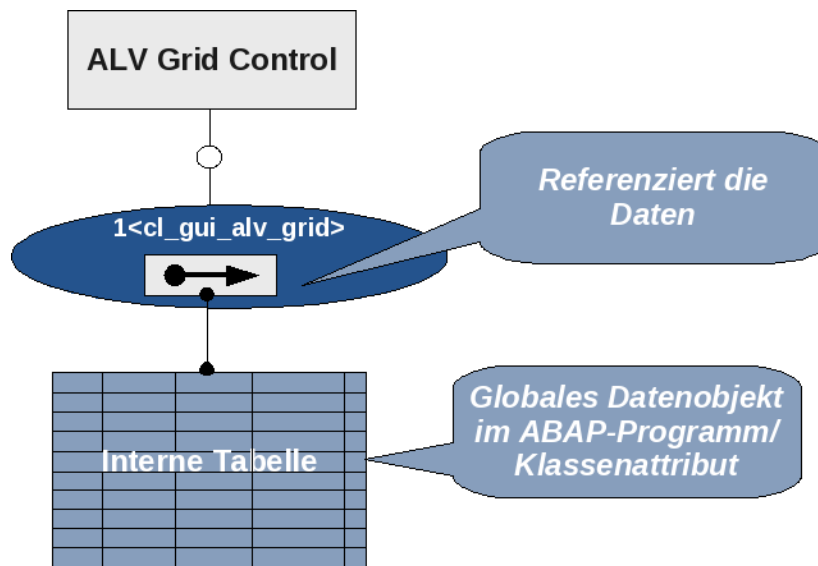


Abbildung 10: Herkunft der Daten eines ALV Grid Control

Neben den eigentlichen Daten werden dem ALV Grid Control (bzw. dem entsprechenden Objekt) auch Informationen zur Darstellung dieser Daten übergeben. Hierzu gibt es verschiedene Möglichkeiten, die sich vom einfachen übergeben des Namens einer Struktur, deren Aufbau für die Darstellung verwendet werden soll, bis hin zu Tabellen (Feldkatalogen) mit einer Konfiguration der einzelnen Spalten erstrecken. Auch die Sortier- und Filterkriterien können vorgegeben werden. Die verschiedenen Möglichkeiten werden später noch näher erläutert. Zunächst aber erfahren Sie, wie Daten überhaupt für die Darstellung bereitgestellt werden.

Für die Übergabe der Daten (bzw. der Referenz auf diese Daten) und der Konfiguration der Darstellung dient die Instanzmethode **set_table_for_first_display** aus der Klasse **cl_gui_alv_grid**. Deren Changing-Parameter **it_outtab** dient der Datenübergabe. Es gibt einige weitere Parameter, die Sie der folgenden Abbildung entnehmen können.

CALL METHOD my_alv_grid->set_table_for_first_display	
* EXPORTING	
* <i>i_structure_name</i> =	Globaler Strukturtyp zu Feldkataloggenerierung
* <i>is_variant</i> =	Verwaltung von Varianten
* <i>i_save</i> =	
* <i>i_default</i> =	
* <i>is_layout</i> =	Struktur für Darstellungsoptionen
* <i>is_print</i> =	Struktur für die Druckausgabe
* <i>it_special_groups</i> =	Interne Tabelle mit Texten für Feld-Gruppen
* <i>it_toolbar_excluding</i> =	Interne Tabelle mit inaktiven Funktionen
CHANGING	
* <i>it_outtab</i> =	Interne Tabelle mit darzustellenden Daten
* <i>it_fieldcatalog</i> =	Interne Tabelle mit Feldkatalog
* <i>it_sort</i> =	Interne Tabelle mit initialen Sortierkriterien
* <i>it_filter</i> =	Interne Tabelle mit initialen Filterkriterien
EXCEPTIONS	
...	

Abbildung 11: Aufruf der Methode set_table_for_first_display

Die einfachste Möglichkeit der Konfiguration der Darstellung ist die bereits erwähnte Übergabe des Namens eines globalen Strukturtyps als Zeichenkette (*i_structure_name*). Aus diesem wird dann automatisch ein Feldkatalog generiert. Ein manuell erzeugter Feldkatalog kann hingegen als *it_fieldcatalog* in Form einer internen Tabelle übergeben werden.

Mit den Parametern *is_layout* bzw. *is_print* können Optionen für die Darstellung bzw. die Druckausgabe übergeben werden. Der Parameter *it_special_groups* dient der Übergabe einer internen Tabelle, die für die Gruppierung von Spalten benötigt wird. Dies ist besonders dann sinnvoll, wenn die Tabelle sehr viele Spalten besitzt und die Darstellung daher unübersichtlich zu werden droht. Die Parameter *is_variant*, *i_save* und *i_default* konfigurieren Varianten, letzterer legt dabei eine Defaultvariante fest.

Mit der Methode *refresh_table_display* des ALV-Objektes kann veranlasst werden, dass die Daten und Darstellungsoptionen erneut von der repräsentativen Instanz an das ALV Grid am Client gesendet werden. Diese Methode wird etwa nach Änderungen der Daten im ABAP-Code aufgerufen, damit diese am Client sichtbar werden. Diese Methode verfügt über zwei optionale Parameter. Mit dem Parameter *i_soft_refresh* kann angegeben werden, dass nur die Daten, nicht aber Einstellungen wie Sortierung, Filter usw. neu gesendet werden sollen. Durch Angabe des Parameters *is_stable* kann hingegen die Scrollposition bezüglich der Zeile und/oder der Spalte beibehalten werden. Der Parameter ist als Struktur des Typs *lvc_s_stbl* zu übergeben. Die Struktur enthält die Komponenten *row* und *col*, über die die Einstellung bezüglich der Zeile bzw. der Spalte vorgenommen werden kann.


```
CALL METHOD my_alv_grid->refresh_table_display
```

```
EXPORTING
```

```
* is_stable =
```

```
* i_soft_refresh =
```

```
DATA <st_name> TYPE lvc_s_stbl.
```


```
<st_name>-row =
```

```
<st_name>-col =
```

Abbildung 12: Aktualisieren der Darstellung

8.2 Praxis: Übung zur Ausgabe von Daten per ALV Grid Control

In dieser Übung werden Sie ihr erstes ALV Grid Control erstellen. Mit einem Dynpro-Programm sollen die im System gespeicherten Fluggesellschaften in einer Tabelle (einem ALV Grid Control) ausgegeben werden. Erstellen Sie dazu ein neues Programm **ZZ_####_ALV** in Ihrem Paket. Es soll sich um ein Programm **mit** TOP-Include handeln. Benennen Sie das TOP-Include **ZZ_####_ALV_TOP**.

Legen Sie in diesem Programm ein Dynpro mit der Nummer 100 an und öffnen Sie den Screen Painter. Erzeugen Sie dort durch Anklicken der Schaltfläche  und markieren des Bereichs auf der Dynpro-Oberfläche einen **Custom Control**-Bereich. Geben Sie diesem den Namen **MY_CUSTOM_CONTROL**. Legen Sie außerdem in der Elementliste einen Namen für das OK-Code-Feld fest. Sichern Sie und verlassen Sie den Screen Painter wieder.

Sorgen Sie als nächstes dafür, dass man das Programm über die Zurück- bzw. Beenden-Schaltflächen wieder verlassen kann. Kommentieren Sie dazu in der Ablauflogik **MODULE STATUS_0100** ein. Legen Sie das Modul per Doppelklick auf den Namen an. Erstellen Sie für das Modul ein neues Include. Achten Sie dabei darauf, dass es sich im Kundennamensraum befindet. Kommentieren Sie im Modul den PF-Status ein, vergeben Sie einen geeigneten Namen und legen Sie ihn per Doppelklick an. Pflegen Sie einen geeigneten Kurztext. Legen Sie dann Funktionscodes für die Schaltflächen fest und sichern Sie den Status. Legen Sie als nächstes ein PAI-Modul an, in welchem Sie den OK-Code abfragen und entsprechend reagieren (vergessen Sie nicht, im TOP-Include die OK-Code-Variable zu definieren).

Speichern, prüfen und aktivieren Sie alle noch nicht aktiven Objekte. Legen Sie eine Transaktion **ZZ_####_ALV** an, mit der das Programm gestartet werden kann, und rufen Sie diese auf. Das Fenster bleibt leer – bislang wurde lediglich der Custom Control-Bereich angelegt. Der Bereich enthält aber noch keine ALV Grid Control-Instanz. Für diese wird wie zuvor dargestellt zunächst ein Container Control benötigt. Definieren Sie die Referenzen für beide Objekte durch entsprechende **DATA**-Anweisungen im TOP-Include. Bezeichnen Sie die Referenz auf den Container mit **r_container** und die Referenz auf das ALV Grid Control mit **r_alv**.

Legen Sie für die Initialisierung der Objekte ein neues PBO-Modul **INIT_ALV** an. Nutzen Sie auch hier zum Anlegen die Vorwärtsnavigation (Doppelklick), nachdem Sie die entsprechende **MODULE**-Zeile in den Dynpro-Code geschrieben haben. Legen Sie für dieses Modul wieder ein neues Include an. Erzeugen Sie im Modul zunächst das Container-Objekt.

Übergeben Sie dem Konstruktor als `container_name` den Namen des Custom Control-Bereichs auf Ihrem Dynpro.

```
CREATE OBJECT r_container
EXPORTING
  container_name = 'MY_CUSTOM_CONTROL'.
```

Abbildung 13: Erstellen des Containerobjekts: SAP-System-Screenshot

Achtung: Es ist ein häufiger Fehler, dass der hier angegebene Containername nicht mit dem Namen des Custom-Control-Bereichs auf dem Dynpro übereinstimmt. Beachten Sie dies insbesondere, wenn Sie den hier gesehenen Code in einem späteren Programm wiederverwenden möchten.

Erzeugen Sie als nächstes das ALV-Objekt. Übergeben Sie dem Konstruktor als `i_parent` die Referenz auf das Containerobjekt. So wird das ALV Grid Control wie gewünscht erstellt. Es zeigt aber noch nichts an, da noch die Daten und die Art ihrer Darstellung angegeben werden müssen. Definieren Sie zunächst eine interne Tabelle vom Typ SCARR im TOP-Include. Die Tabelle ist so global im gesamten Programm sichtbar. Holen Sie in Ihrem PBO-Modul **INIT_ALV** die entsprechenden Daten aus der Datenbank und schreiben Sie diese in die interne Tabelle.

Rufen Sie anschließend die Methode `set_table_for_first_display` des ALV-Objekts auf. Verwenden Sie hierzu die **Muster**-Schaltfläche. Wählen Sie dort Muster zu ABAP Objects und geben Sie auf dem folgenden Fenster die benötigten Angaben für den Bereich **Call Method** an.

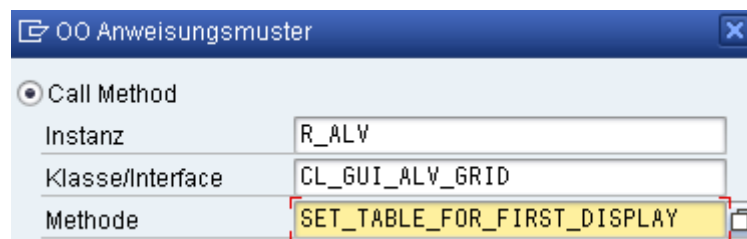
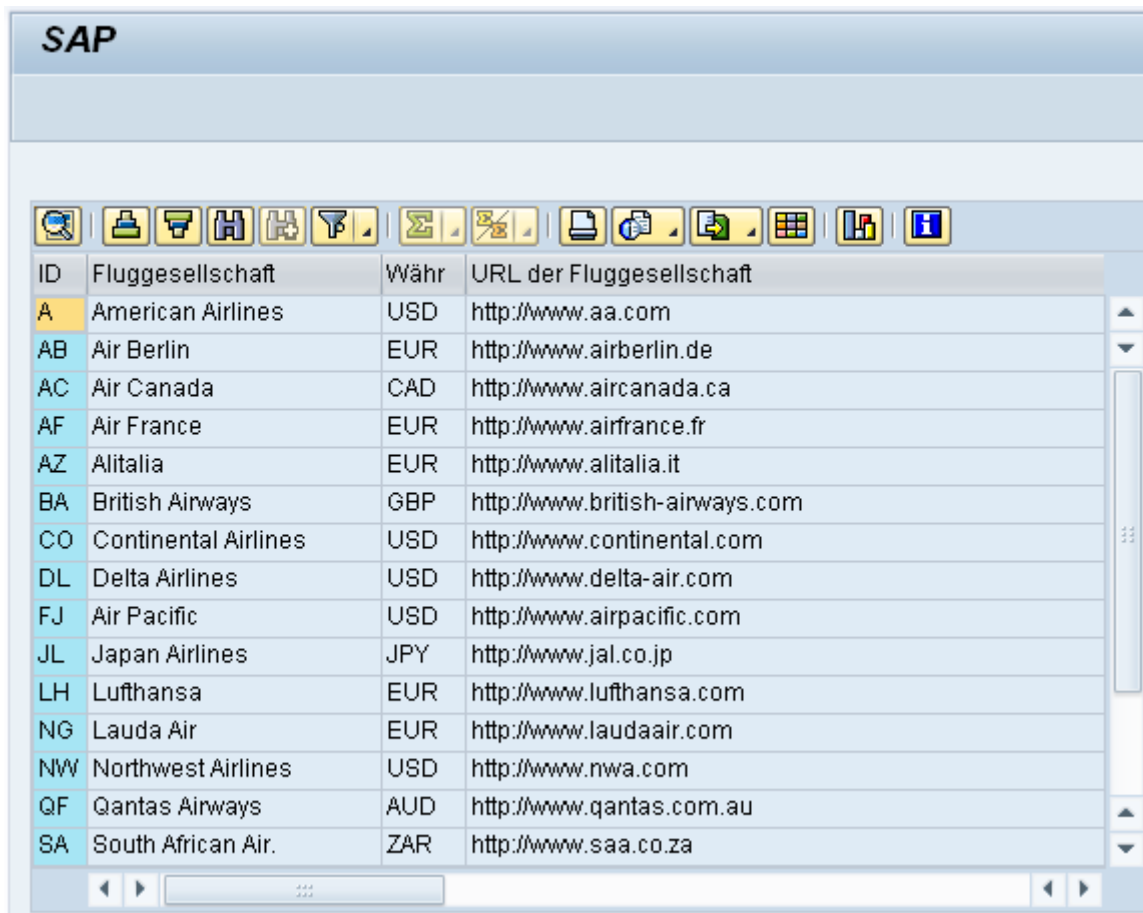


Abbildung 14: Muster zum Aufruf einer Methode: SAP-System-Screenshot

Der Feldkatalog soll in diesem Programm automatisch vom System erstellt werden. Übergeben Sie daher die Datenbankstruktur, die dieser Erstellung zugrunde gelegt werden soll (SCARR), an den Parameter `i_structure_name` (in Hochkommata; stellen Sie sicher, dass der Parameter und auch das `EXPORTING`-Schlüsselwort nicht auskommentiert sind) und an den Changing-Parameter `it_outtab` die im TOP-Include definierte interne Tabelle. Alle optionalen Parameter werden beim Einfügen von Code über die Muster-Schaltfläche zunächst einkommentiert. Beachten Sie jedoch, dass ein Aufruf der Methode nur mit der Übergabe der Datentabelle nicht funktionieren würde – eine der Möglichkeiten zum Festlegen des Feldkatalogs muss genutzt werden, auch wenn beide für sich genommen optional sind. Speichern und prüfen Sie das Include und aktivieren Sie anschließend alle noch nicht aktivierten Objekte. Rufen Sie dann erneut Ihre für dieses Programm erstellte Transaktion auf. Die Programmoberfläche enthält nun ein ALV Grid Control in dem die Daten aller Fluggesellschaften dargestellt werden.



The screenshot shows the SAP interface with an ALV Grid Control table. The table has four columns: ID, Fluggesellschaft, Währ, and URL der Fluggesellschaft. The data is as follows:

ID	Fluggesellschaft	Währ	URL der Fluggesellschaft
A	American Airlines	USD	http://www.aa.com
AB	Air Berlin	EUR	http://www.airberlin.de
AC	Air Canada	CAD	http://www.aircanada.ca
AF	Air France	EUR	http://www.airfrance.fr
AZ	Alitalia	EUR	http://www.alitalia.it
BA	British Airways	GBP	http://www.british-airways.com
CO	Continental Airlines	USD	http://www.continental.com
DL	Delta Airlines	USD	http://www.delta-air.com
FJ	Air Pacific	USD	http://www.airpacific.com
JL	Japan Airlines	JPY	http://www.jal.co.jp
LH	Lufthansa	EUR	http://www.lufthansa.com
NG	Lauda Air	EUR	http://www.laudaair.com
NW	Northwest Airlines	USD	http://www.nwa.com
QF	Qantas Airways	AUD	http://www.qantas.com.au
SA	South African Air.	ZAR	http://www.saa.co.za

Abbildung 15: Ihr erstes ALV Grid Control: SAP-System-Screenshot

Sie haben nun die Schritte zum Erstellen eines ALV Grid Control und zur Anzeige von Daten in diesem Element am System nachvollzogen.

8.3 Eigene Feldkataloge

In der Übung haben Sie den Feldkatalog automatisch generieren lassen, indem Sie einen Strukturtyp aus dem Dictionary übergeben haben. In diesem Unterkapitel wird erläutert, wie Sie selbst einen Feldkatalog erstellen und mit diesem das Erscheinungsbild der Tabelle verändern können.

Der Feldkatalog ist eine interne Tabelle, die Sie im ABAP-Programm definieren. Sie wird über den Tabellentyp LVC_T_FCMT typisiert, ihr Zeilentyp ist LVC_S_FCMT. Mit der Tabelle können die Einstellungen für die Spalten komplett selbst vorgenommen oder die Spalten aus dem übergebenen Strukturtypen ergänzt bzw. deren Darstellung überschrieben werden. Die einzelnen Zeilen der Tabelle beschreiben die Spalten, die im ALV Grid Control angezeigt werden sollen.

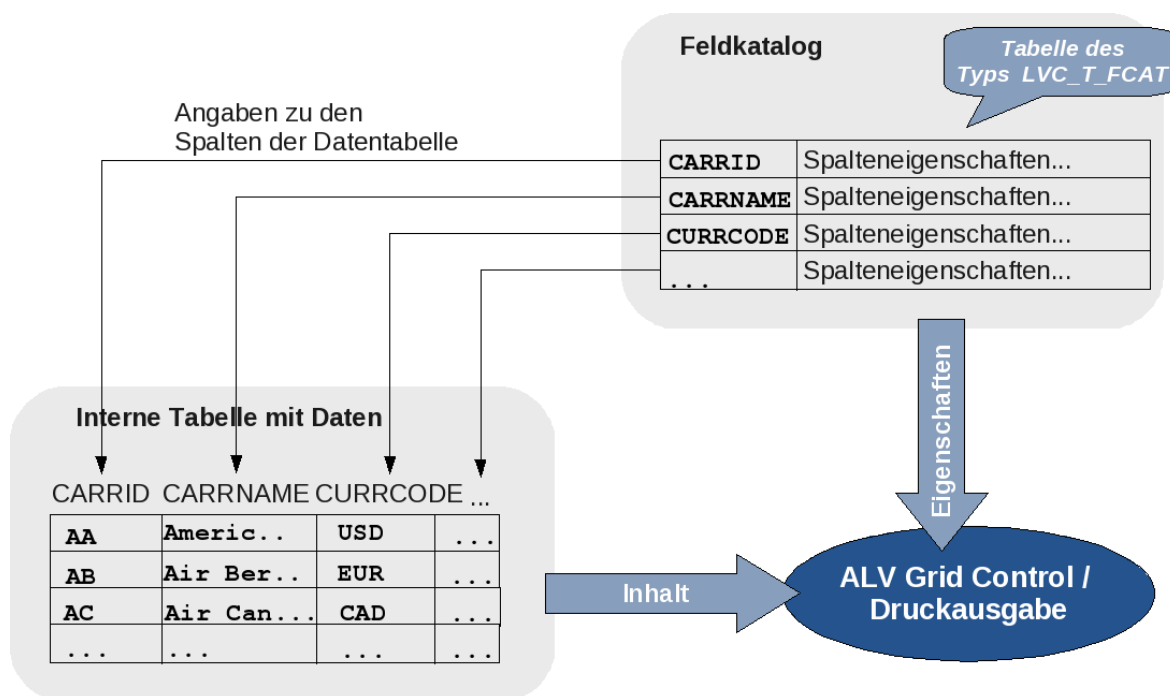


Abbildung 16: Zusammenhang zwischen Datentabelle und Feldkatalog

Wenn eine Struktur zur Generierung eines Feldkatalogs an das ALV Grid Control übergeben wird, kann durch eine zusätzliche Feldkatalog-Tabelle die Felddescription einzelner Felder übersteuert werden. Ebenfalls kann die Beschreibung ergänzt werden, wenn nicht alle Felder der Datentabelle auch in der Struktur, die für die automatische Feldkatalog-Generierung übergeben wurde, vorhanden sind. Die folgende Abbildung veranschaulicht den Zusammenhang:

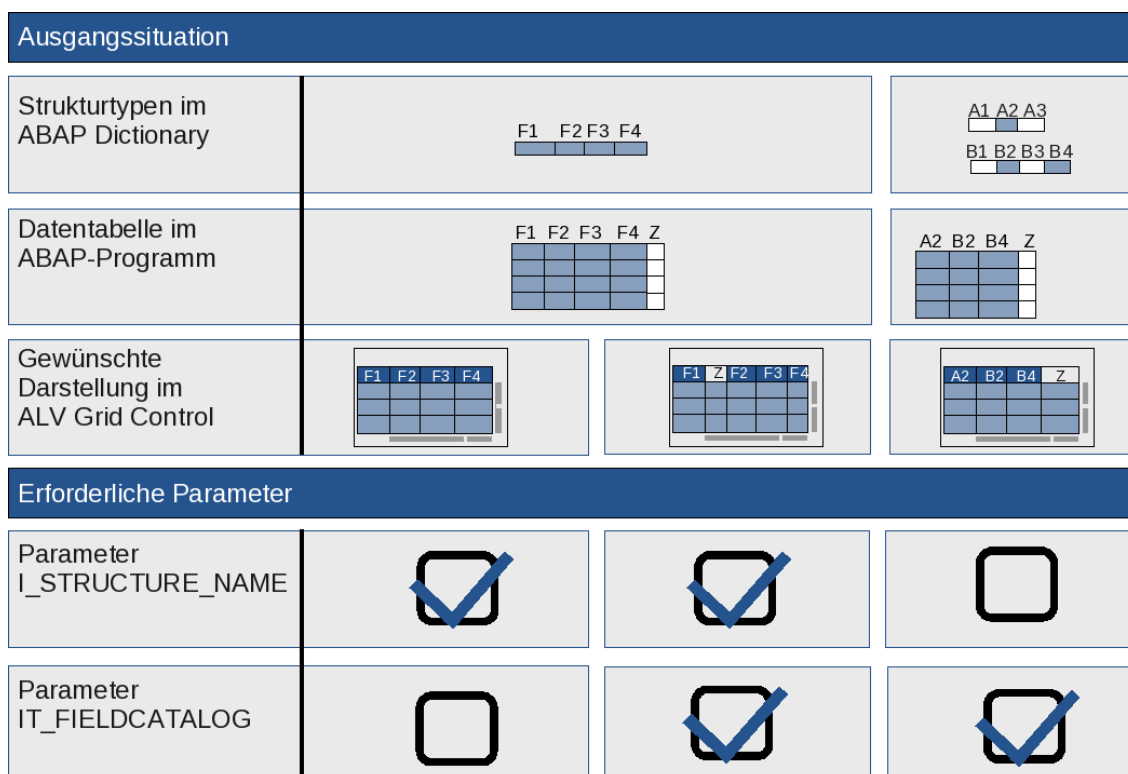


Abbildung 17: Erforderliche Parameter zur Feldkatalog-Erstellung

Der Aufbau des Feldkatalogs, also der internen Tabelle, die als Parameter übergeben wird, besteht aus den Feldern **FIELDNAME**, **REF_FIELD**, **REF_TABLE**, sowie diversen weiteren Feldern zur Festlegung der Eigenschaften der Spalten des ALV Grid Control. Das Feld **FIELDNAME** muss immer angegeben werden, es enthält den Namen der Spalte, die beschrieben werden soll. Um bei der Beschreibung auf Eigenschaften aus dem Dictionary zuzugreifen, kann **REF_TABLE** angegeben werden. Weicht der Feldname im Dictionary vom Spaltennamen ab, muss der Feldname des Dictionary als **REF_FIELD** angegeben werden. Wird kein Bezug zum Dictionary hergestellt, müssen alle Einstellungen manuell über die weiteren Felder des Feldkatalogs vorgenommen werden. Auch bei Bezug auf das Dictionary können die weiteren Felder benutzt werden. Die Angaben übersteuern dann die Angaben aus dem Dictionary. Die folgende Abbildung zeigt ein Beispiel:

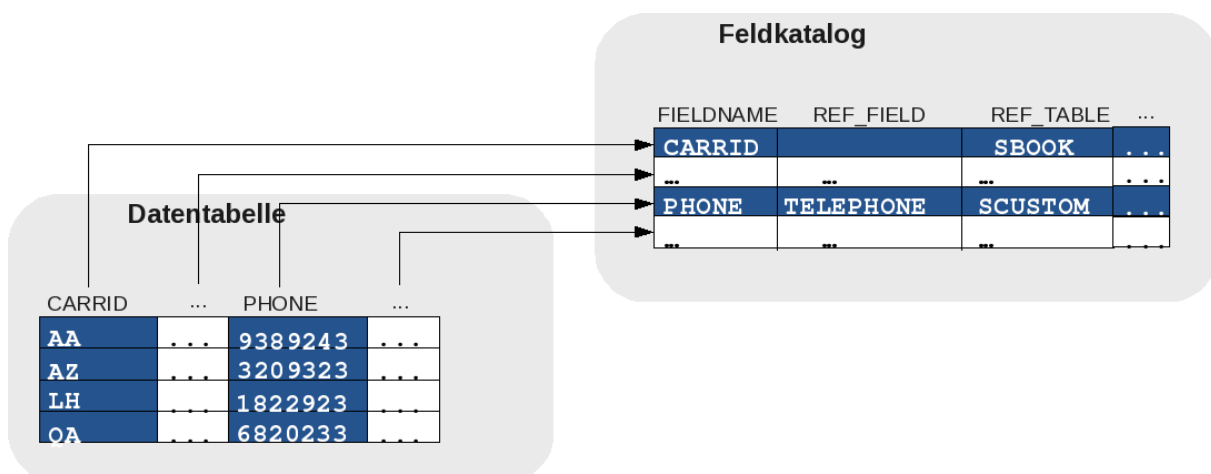


Abbildung 18: Beispieleinträge in einem Feldkatalog

Das Feld **CARRID** kommt in der Tabelle **SBOOK**, die als **REF_TABLE** angegeben ist, unter demselben Namen vor. Daher ist hier das Feld **REF_FIELD** nicht anzugeben. Die Spalte für die Telefonnummer hat hingegen in der Datentabelle einen von der als **REF_TABLE** angegebenen Tabelle **SCUSTOM** abweichenden Namen, so dass das Feld **REF_FIELD** hier angegeben werden muss.

8.3.1 Angaben für nicht-Strukturfelder im Feldkatalog

In den weiteren Spalten des Feldkatalogs können Angaben gemacht werden, die ein Element beschreiben, das nicht aus einer Struktur des Dictionary kommt. Dies umfasst die folgenden Angaben:

Tabelle 1: Beschreibung von Spalten ohne Bezug auf eine Dictionary-Struktur

Feldkatalog-Spalte	Bedeutung
ROLLNAME	Name eines Datenelements aus dem Dictionary
INTLEN	Interne Spaltenbreite (nur für Conversion Exits)
DD_OUTLEN	Breite der Spalte bei Ausgabe (nur für Conversion Exits)
INTTYPE	ABAP-Datentyp (z. B. c, i oder n)

8.3.2 Texte im Feldkatalog

Bei Spalten, für die kein Dictionary-Bezug hergestellt wurde oder die dort festgelegten Eigenschaften überschrieben werden sollen, können auch die Texte zur Spaltenbeschreibung angegeben werden. Hierfür werden die folgenden Feldkatalog-Spalten benutzt:

Tabelle 2: Texte mit dem Feldkatalog festlegen

Feldkatalog-Spalte	Bedeutung
COLDDICTXT	Angabe, welcher der im Dictionary oder in den folgenden Feldern definierten Texten für die Spaltenüberschrift verwendet werden soll. 'S', 'M', 'L' oder 'R' rufen den kurzen, mittleren, langen bzw. Überschrifts-Text auf, eine Angabe von ' ' (Leerzeichen) führt hingegen zur Verwendung des Texts aus der Feldkatalog-Spalte COLTEXT.
SCRTEXT_S	Kurzer Text (Überlagert ggf. entsprechenden Dictionary-Text)
SCRTEXT_M	Mittlerer Text (Überlagert ggf. entsprechenden Dictionary-Text)
SCRTEXT_L	Langer Text (Überlagert ggf. entsprechenden Dictionary-Text)
REPTXT	Überschrift (Überlagert ggf. entsprechenden Dictionary-Text)
COLTEXT	Frei definierter Text für die Überschrift (wenn COLDDICTXT=' ')
SELTEXT	Frei definierter Selektionstext
TOOLTIP	Frei definierter Text für den Tooltip (Anzeige bei Positionierung des Maus über der Spaltenüberschrift)

Sind weder COLDDICTXT noch COLTEXT gesetzt, bestimmt das System anhand der Spaltenbreite einen geeigneten Text für die Spaltenüberschrift aus den vier üblichen Texten des Datenelements. Für den Selektionstext wird, wenn das SELTEXT-Feld nicht gesetzt wird, der längste dieser vier Texte verwendet. Sind diese nicht gesetzt, wird stattdessen das Tooltip-Feld verwendet, ist auch dies nicht gesetzt, kommt der Name des Felds selbst zum Einsatz.

8.3.3 Währungs- und Mengenangaben im Feldkatalog

Felder, die Währungsangaben oder Angaben in bestimmten Einheiten enthalten, benötigen eine Festlegung der zugrundeliegenden Einheit. Hier gibt es jeweils zwei Möglichkeiten. Entweder die Einheit wird für die gesamte Spalte festgelegt, oder es wird eine Spalte angegeben, die für jede Zeile der Datentabelle die zu verwendende Einheit enthält.

Tabelle 3: Feldkatalogspalten für Währungs- und Mengenangaben

Feldkatalog-Spalte	Bedeutung
CFIELDNAME	Name der Spalte, die die Währungsangabe zur aktuellen Spalte enthält
CURRENCY	Währung, die für alle Einträge in dieser Spalte gelten soll
QFIELDNAME	Name der Spalte, die die Einheit zur aktuellen Spalte enthält
QUANTITY	Einheit, die für alle Einträge in dieser Spalte gelten soll

Die Felder CURRENCY bzw. QUANTITY haben dabei eine höhere Priorität als die Felder CFIELDNAME bzw. QFIELDNAME, d. h. eine dort gemachte Angabe führt dazu dass die Angabe im jeweils anderen Feld ignoriert wird.

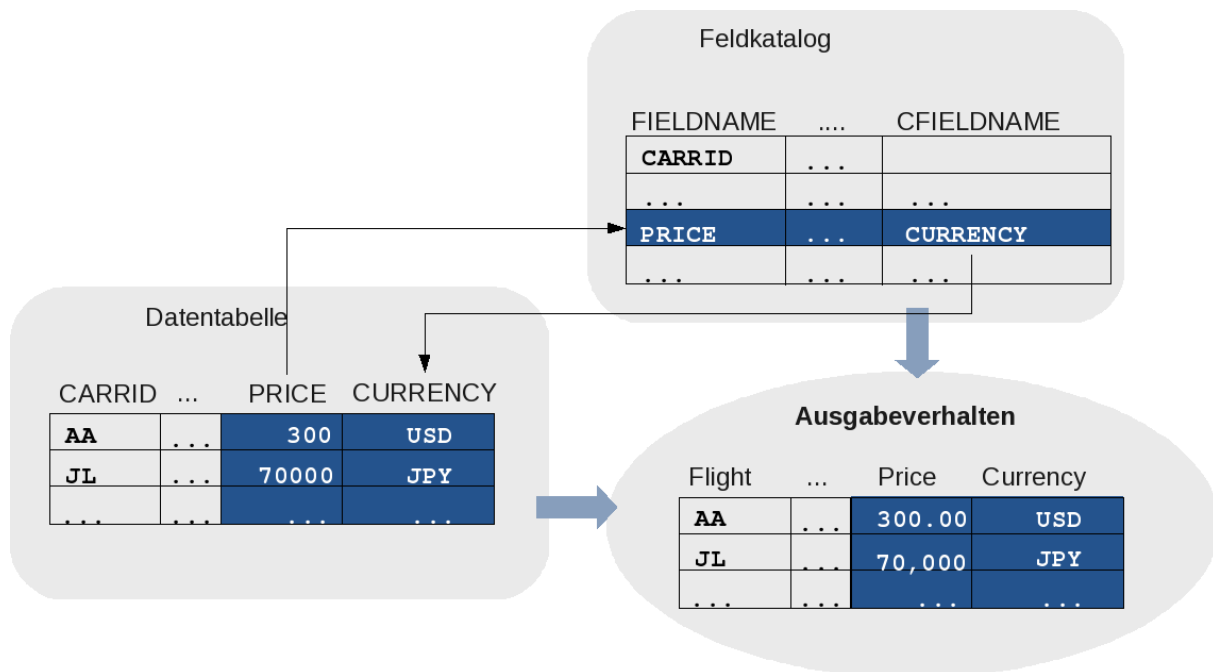


Abbildung 19: Formatierung von Währungsfeldern

Die obige Abbildung zeigt ein Beispiel für die Verwendung der Währungsformatierung. Im Feldkatalog existiert ein Eintrag für die Spalte PRICE. In diesem Eintrag ist als CFIELDNAME der Wert CURRENCY eingetragen. Das System verwendet daher die Währungsangabe aus Spalte der mit diesem Namen in der Datentabelle. Im ersten Datensatz ist dies die Währung USD, im zweiten Datensatz die Währung JPY. Die Formatierung unterscheidet sich hier bezüglich des Trennzeichens und der Anzahl von Nachkommastellen.

8.3.4 Weitere Angaben im Feldkatalog

Die Ausgabeeigenschaften können über eine Vielzahl weiterer Spalten des Feldkatalogs angepasst werden. Die folgende Tabelle gibt Ihnen einen Überblick über die Möglichkeiten.

Tabelle 4: Ausgabe- und Formateigenschaften im Feldkatalog

Feldkatalog-Spalte	Bedeutung
CHECKBOX	Gibt an ob das Feld als (nicht eingabebereite) Checkbox dargestellt werden soll.
COL_POS	Gibt die Position der Spalte an. Bei der Positionierung sind ausgeblendete Spalten mitzuzählen. Wird keine Positionierung angegeben, folgen die manuell über den Feldkatalog beschriebenen Felder auf die anderen Felder der Datentabelle.
DO_SUM	Gibt an, ob eine Gesamtsumme der Spalte angezeigt werden soll.
HOTSPOT	Gibt an, ob die Spalte als Hotspot fungieren soll.
NO_SUM	Gibt an, ob die Verwendung von Aggregatfunktionen für die Spalte unterbunden werden soll.
EMPHASIZE	Gibt an ob die Spalte hervorgehoben werden soll. Der Wert 'X' sorgt für eine Standardhervorhebung. Alternativ kann das Schema 'C<Farbkonstante><intensiviert><invertiert>' verwendet werden, wobei die Farbkonstanten der Typgruppe col entnommen werden können. Die Angaben intensiviert bzw. invertiert sind über die Ziffern 0 bzw. 1 anzugeben.
KEY	Gibt an ob es sich um ein Schlüsselfeld handelt.
LOWERCASE	Gibt an, ob zwischen Groß- und Kleinschreibung unterschieden wird.

NO_OUT	Blendet die Spalte an der Oberfläche aus (sie ist jedoch in Funktionen wie der Sortierung weiterhin auswählbar)
OUTPUTLEN	Gibt die Spaltenbreite an.
TECH	Gibt an, ob es sich um eine technische Spalte handelt. Technische Spalten tauchen weder in der Liste noch in Dialogen zur Feldauswahl auf.
DECIMALS_O	Gibt an, wie viele Dezimalstellen ausgegeben werden sollen
DECIMALFIELD	Gibt alternativ zu DECIMALS_O an, in welcher Spalte die Anzahl der Dezimalstellen jeder einzelnen Zelle der Spalte stehen.
EDIT_MASK	Ermöglicht die Angabe einer Eingabemaske zur Formatierung (analog zum Zusatz USING EDIT MASK der WRITE-Anweisung)
EXPONENT	Gibt einen festen Exponenten zur Darstellung von Fließkommazahlen an.
ICON	Gibt an, ob die Spalte als Icon dargestellt werden soll. Mögliche Icons können mit dem Programm SHOWICON angezeigt werden.
SYMBOL	Gibt an, ob die Spalte als Symbol dargestellt werden soll. Mögliche Symbole können mit dem Programm SHOWSYMB angezeigt werden.
LZERO	Gibt an, ob bei numerischen Zeichenketten führende Nullen dargestellt werden sollen.
JUST	Ermöglicht die Ausrichtung von Zeichenketten. Mögliche Werte sind ' ', 'L', 'R' und 'C'.
NO_SIGN	Ermöglicht das Unterdrücken des Minuszeichens bei negativen Zahlen.
NO_ZERO	Ermöglicht es, Felder mit dem Wert 0 leer anzuzeigen.
ROUND	Ermöglicht die Rundung von Zahlen analog zum Zusatz ROUND der WRITE-Anweisung.
ROUNDFIELD	Ermöglicht die Angabe einer Spalte der Datentabelle, in der zu jeder Zeile die Rundungsangabe zu finden ist.

Wenn eine Tabelle sehr viele Spalten enthält, ist es für den Anwender hilfreich, diese Spalten zu gruppieren. Der Anwender hat dann die Möglichkeit, die Spaltenliste nach der Gruppe zu filtern und die Oberfläche gewinnt somit an Übersichtlichkeit. Hierfür wird im Feld SP_GOUP eine Gruppe zugeordnet. Ihren Namen erhält die Gruppe, indem der Methode **set_table_for_first_display** der Parameter IT_SPECIAL_GROUPS übergeben wird. Wenn Zwischensummen über ein Feld der Datentabelle gebildet werden sollen, und ein anderes Feld dessen Beschreibung enthält, kann dieses andere Feld über TXT_FIELD angegeben werden. Bei der Zwischensumme wird dann dieses Feld zusätzlich ausgegeben, also bspw. ein Artikelname zusätzlich zur Artikelnummer.

8.4 Praxis: Übung zum Feldkatalog

In dieser Übung werden Sie ein ALV Grid Control durch einen Feldkatalog mit Spaltenbeschreibungen versehen. Erstellen Sie hierzu ein neues Programm **ZZ_####_FCAT** und wählen Sie **mit TOP-Include** aus. Der Name des TOP-Includes soll **ZZ_####_FCAT_TOP** lauten.

Das Programm soll aus drei Dynpros bestehen. Legen Sie diese mit den Nummern 100, 200 und 300 an. Mit dem Programm sollen zu einem Reisebüro die dort getätigten Buchungen angezeigt werden können. Legen Sie daher im TOP-Include einen Schnittstellenarbeitsbereich für Reisebüros an:

```
TABLES stravelag.
```

Definieren Sie dort außerdem eine Variable für den OK-Code. Öffnen Sie dann das Layout von Dynpro 100, und fügen Sie durch Übernahme aus dem Dictionary (aus dem **Dict./Programmfelder**-Fenster des Graphical Screen Painters) ein Eingabefeld für die Nummer (**agencynum**) des Reisebüros ein. Ergänzen Sie eine Schaltfläche zum Bestätigen der Eingabe mit dem Funktionscode **OK** und setzen Sie den Namen des OK-Code-Feldes in der Elementliste. Sichern Sie das Dynpro. Kommentieren Sie in der Dynpro-Ablauflogik den Aufruf des Moduls STATUS_0100 ein und legen Sie es an. Sorgen Sie dafür, dass das Programm einen Titel sowie eine Zurück-Funktion besitzt, indem Sie die auskommentierten Zeilen verwenden und implementieren Sie anschließend die Verarbeitung des OK-Codes in einem PAI-Modul, so dass das Programm verlassen werden kann und das Anklicken des Buttons zu Dynpro 200 führt. Legen Sie für die Module in dieser Aufgabe jeweils Includes an, deren Namen mit **ZZ_####_** beginnen.

Öffnen Sie als nächstes Dynpro 200. Legen Sie dort einen Custom Control-Bereich mit dem Namen **MY_AREA** an, in dem später die Liste der Buchungen erscheinen soll. Setzen Sie auch hier den Namen des OK-Code-Felds und sichern Sie. Verlassen Sie anschließend den Screen Painter. Legen Sie im TOP-Include die Referenzvariablen **r_container** für das Container Control und **r_alv** für das ALV Grid Control an. Erstellen Sie dann ein Modul im PBO von Dynpro 200, in dem Sie die entsprechenden Objekte erzeugen.

Es fehlen nun noch die Daten sowie der Feldkatalog zu deren Beschreibung. Definieren Sie dafür im TOP-Include einen lokalen Strukturtypen **ty_book**. Dieser soll Felder für den Namen, Ort und Land des Kunden, die wie in der Tabelle SCUSTOM typisiert sind, sowie alle Felder der SBOOK-Tabelle enthalten. Nutzen Sie hierfür den **INCLUDE TYPE**-Befehl (Achtung! Benutzen Sie **INCLUDE TYPE** nicht innerhalb des Kettensatzes, andernfalls wird das System die Zeile so interpretieren als wollten Sie **ein Feld** mit dem Namen "INCLUDE" definieren). Definieren Sie anschließend eine interne Tabelle **it_book** für die Buchungen sowie eine interne Tabelle für den Feldkatalog. Lesen Sie im PBO von Dynpro 200 die Buchungsdaten in die Tabelle ein. Sie benötigen hierfür einen Join der Buchungs- und Kundentabelle. Vergessen Sie nicht den **WHERE**-Teil, in dem sie die Reisebüronummer mit der entsprechenden Komponente des Schnittstellenarbeitsbereichs vergleichen müssen den sie in Dynpro 100 verwendet haben.

Erstellen Sie als nächstes den Feldkatalog. Definieren Sie dafür einen Arbeitsbereich **wa** des Typs **LVC_S_FCAT**. Fügen Sie mit diesem Einträge für die zusätzlichen Spalten (die nicht Teil der Buchungstabelle sind) in den Feldkatalog (die zuvor definierte interne Tabelle) ein. Beispiel:

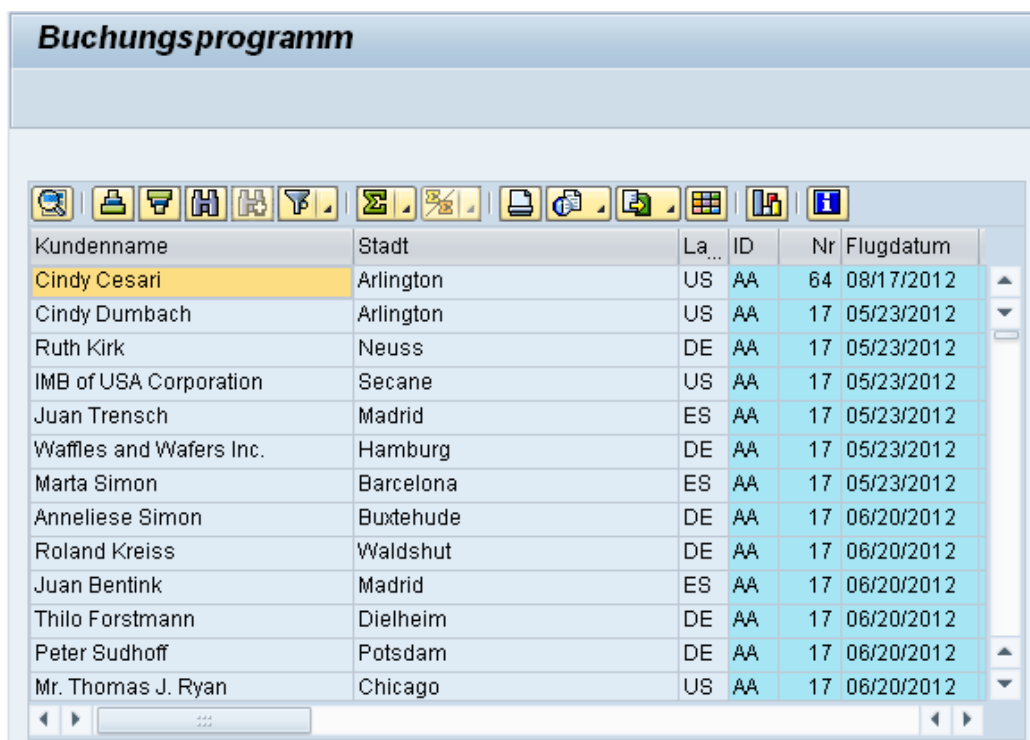
```
wa-fieldname = 'NAME'.  
wa-ref_field = 'NAME'.  
wa-ref_table = 'SCUSTOM'.  
INSERT wa INTO TABLE fkat.
```

Achten Sie darauf, die Zeichenketten in Großbuchstaben zu schreiben. Übergeben Sie den Feldkatalog beim Aufruf der Methode **set_table_for_first_display** (erzeugen Sie den Methodenaufruf mit dem Muster-Button und passen Sie dann die Parameterliste an). Geben Sie dort außerdem **'SBOOK'** als Strukturnamen an, damit die nicht in ihrem Feldkatalog manuell angelegten Einträge automatisch generiert werden, und übergeben Sie die interne Tabelle mit den Daten an den Parameter **it_outtab**.

Stellen Sie sicher, dass die Methode nur beim ersten Aufruf des Dynpro aufgerufen wird. Das bedeutet, das ALV Grid-Objekt soll nur angelegt werden wenn es zuvor initial war und nur dann die Methode aufgerufen werden. Sorgen Sie dafür, dass bei allen weiteren Aufrufen (mit nicht-initialem ALV) nur die Daten aktualisiert werden und mit der Methode **refresh_table_display** des ALV Grid Control zum Client gesendet werden.

Implementieren Sie anschließend ein PAI-Modul, in dem Sie den OK-Code verarbeiten und so dafür sorgen, dass ein Klick auf den Zurück-Pfeil wieder zum ersten Dynpro führt.

Speichern, prüfen und aktivieren Sie alle zu aktivierenden Objekte. Legen Sie eine Transaktion **ZZ_####_FCAT** an, mit der Sie das Programm starten können, und führen Sie diese aus. Testen Sie, ob die drei zusätzlichen Spalten und die Spalten von SBOOK sichtbar sind, und ob das Verhalten beim Zurückspringen zum ersten Dynpro und auswählen eines anderen Reisebüros erwartungsgemäß ist.

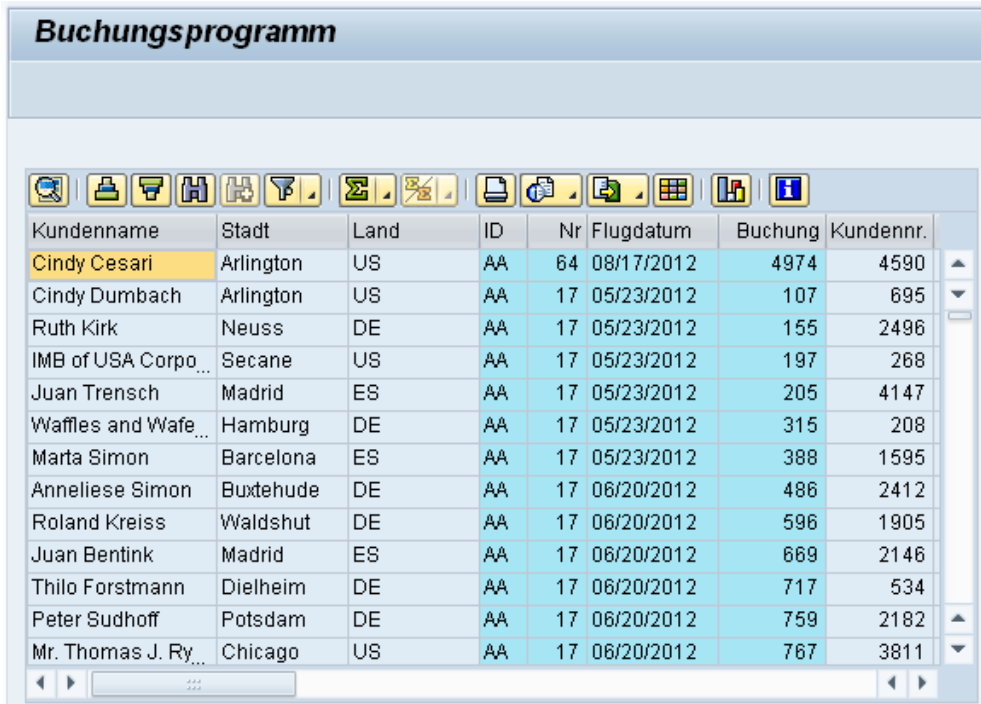


Kundenname	Stadt	La...	ID	Nr	Flugdatum
Cindy Cesari	Arlington	US	AA	64	08/17/2012
Cindy Dumbach	Arlington	US	AA	17	05/23/2012
Ruth Kirk	Neuss	DE	AA	17	05/23/2012
IMB of USA Corporation	Secane	US	AA	17	05/23/2012
Juan Trench	Madrid	ES	AA	17	05/23/2012
Waffles and Wafers Inc.	Hamburg	DE	AA	17	05/23/2012
Marta Simon	Barcelona	ES	AA	17	05/23/2012
Anneliese Simon	Buxtehude	DE	AA	17	06/20/2012
Roland Kreiss	Waldshut	DE	AA	17	06/20/2012
Juan Bentink	Madrid	ES	AA	17	06/20/2012
Thilo Forstmann	Dielheim	DE	AA	17	06/20/2012
Peter Sudhoff	Potsdam	DE	AA	17	06/20/2012
Mr. Thomas J. Ryan	Chicago	US	AA	17	06/20/2012

Abbildung 20: Oberfläche des Programms: SAP-System-Screenshot

In der Ausgabe sehen Sie, dass die Spalten für den Kundennamen und die Stadt sehr breit sind. Überarbeiten Sie ihren Feldkatalog, so dass die Breite dieser Spalten auf 15 bzw. 10 Zeichen reduziert wird. Die Ausgabe sollte danach etwa wie folgt aussehen:

Buchungsprogramm



Kundenname	Stadt	Land	ID	Nr	Flugdatum	Buchung	Kundennr.
Cindy Cesari	Arlington	US	AA	64	08/17/2012	4974	4590
Cindy Dumbach	Arlington	US	AA	17	05/23/2012	107	695
Ruth Kirk	Neuss	DE	AA	17	05/23/2012	155	2496
IMB of USA Corpo...	Secane	US	AA	17	05/23/2012	197	268
Juan Trench	Madrid	ES	AA	17	05/23/2012	205	4147
Waffles and Wafe...	Hamburg	DE	AA	17	05/23/2012	315	208
Marta Simon	Barcelona	ES	AA	17	05/23/2012	388	1595
Anneliese Simon	Buxtehude	DE	AA	17	06/20/2012	486	2412
Roland Kreiss	Waldshut	DE	AA	17	06/20/2012	596	1905
Juan Bentink	Madrid	ES	AA	17	06/20/2012	669	2146
Thilo Forstmann	Dielheim	DE	AA	17	06/20/2012	717	534
Peter Sudhoff	Potsdam	DE	AA	17	06/20/2012	759	2182
Mr. Thomas J. Ry...	Chicago	US	AA	17	06/20/2012	767	3811

Abbildung 21: Verringerte Spaltenbreiten: SAP-System-Screenshot

Achtung: Ein häufiger Fehler in diesem Programm besteht darin, dass die Daten nur beim ersten Aufruf des zweiten Dynpros geladen werden. Testen Sie dieses folgendermaßen:

- Starten Sie das Programm
- Geben Sie das Reisebüro 55 an.
- Schauen Sie sich die Daten auf dem zweiten Dynpro an
- Gehen Sie mit F3 zurück zum ersten Dynpro
- Geben Sie das Reisebüro 123 an.
- Schauen Sie sich die Daten auf dem zweiten Dynpro an. Diese müssen sich von den Daten aus Schritt 3 unterscheiden. Wenn sie dies nicht tun, prüfen Sie, ob Sie vor dem Auffrischen des ALV auch die Daten neu aus der Datenbank lesen.

8.5 Ereignisse des ALV Grid Control

Durch Benutzeraktionen auf dem ALV Grid Control werden Ereignisse ausgelöst, auf die entsprechend reagiert werden kann. Die Umsetzung dessen ist aufgrund der Architektur nicht trivial, da stets zwei Instanzen in Erscheinung treten: Eine Instanz des Controls auf Client-Seite sowie eine repräsentative Instanz auf Seiten des Anwendungsservers, mit der das ABAP-Programm kommuniziert. Glücklicherweise muss sich der Entwickler mit dieser Problematik nicht auseinandersetzen, denn durch das Control Framework wird die Kommunikation zwischen den Instanzen so realisiert, dass die Beschränkung auf die Kommunikation mit der repräsentativen Instanz kein Problem darstellt.

Das vom Benutzer ausgelöste Ereignis wird vom **Automation Controller** im SAP GUI entgegengenommen. Dieser leitet die Information über das Ereignis als speziellen Funktionscode an die Basisdienste des Laufzeitsystems, welche sie wiederum an die repräsentative Instanz leiten, die das Ereignis als Ereignis im Sinne von ABAP Objects auslöst. Dieses Ereignis kann dann entsprechend behandelt werden.

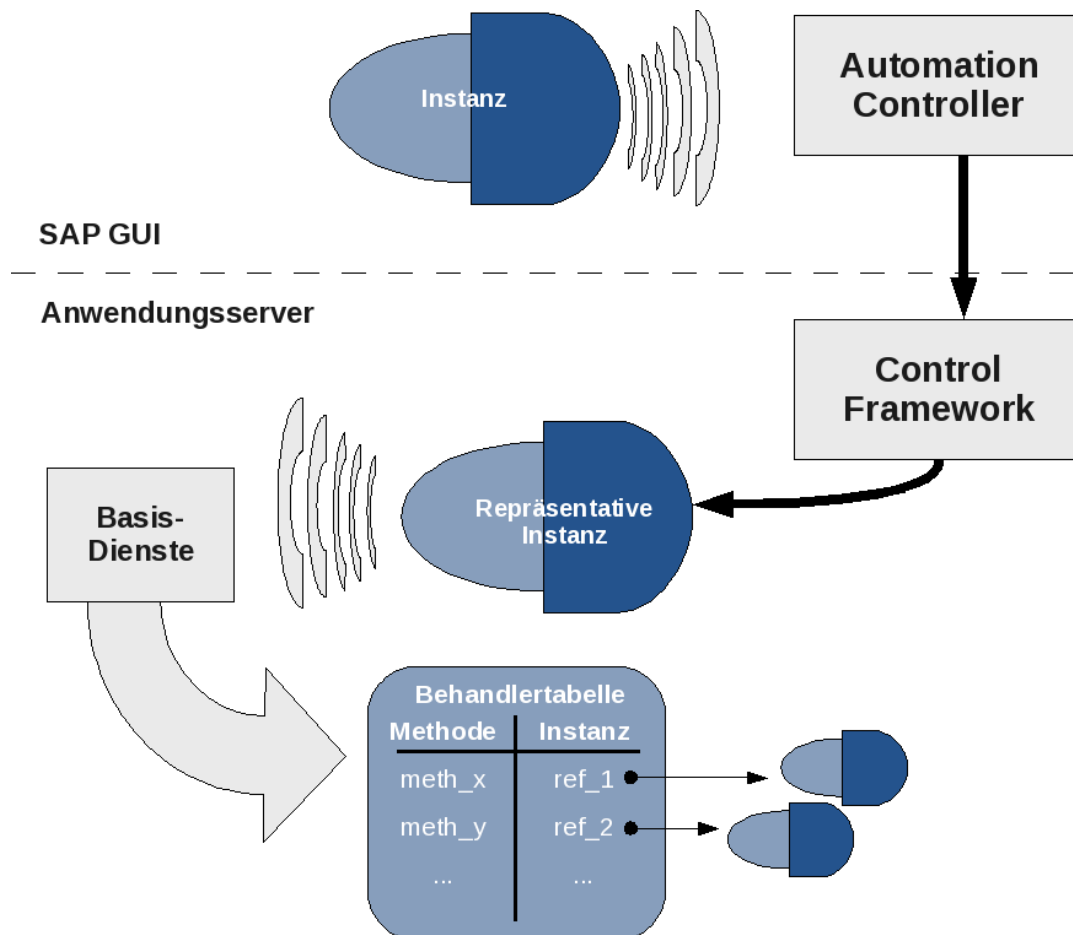


Abbildung 22: Behandlung von Ereignissen des ALV Grid Control

Die Behandlung der Ereignisse erfolgt wie gewohnt durch Registrierung der Behandler für das Ereignis (lesen Sie das Verfahren bei Bedarf im entsprechenden Kapitel des Kurses nach).

Die Klasse **cl_gui_alv_grid** besitzt eine Vielzahl von Ereignissen. Besonders wichtige Ereignisse sind **DOUBLE_CLICK** (Ereignis bei Doppelklick auf eine Zelle der Tabelle), **HOTSPOT_CLICK** (Ereignis bei Klick auf ein Feld einer Hotspot-Spalte) und **USER_COMMAND**. Letzteres wird ausgelöst, wenn der Anwender auf eine von Ihnen definierte Zusatzfunktion des ALV Grid Controls zugreift.

Die Ereignisse besitzen Parameter, um dem Behandler Informationen zu übermitteln. Bei den Klick-Ereignissen sind dies etwa Angaben zur Zeile und Spalte, in der der Klick bzw. Doppelklick erfolgte. Leider ist die Schnittstelle nicht ganz einheitlich:

- Beim Ereignis **HOTSPOT_CLICK** erhalten Sie die Zeilennummer der Datentabelle, die dem angeklickten Eintrag des ALV Grid Controls entspricht, über die Komponente **ROW_ID** des Parameters **ES_ROW_NO**. Dieser ist über den Strukturtyp **LVC_S_ROID** typisiert. Den Namen der Spalte der Datentabelle, die der angeklickten Spalte des ALV Grid Control entspricht, steht in der Komponente **FIELDNAME** des Parameters **E_COLUMN_ID**, der mit dem Strukturtypen **LVC_S_COL** typisiert ist.
- Beim Ereignis **DOUBLE_CLICK** verhält es sich genauso, jedoch heißt hier der Parameter für die Spalte nicht **E_COLUMN_ID** sondern nur **E_COLUMN**.

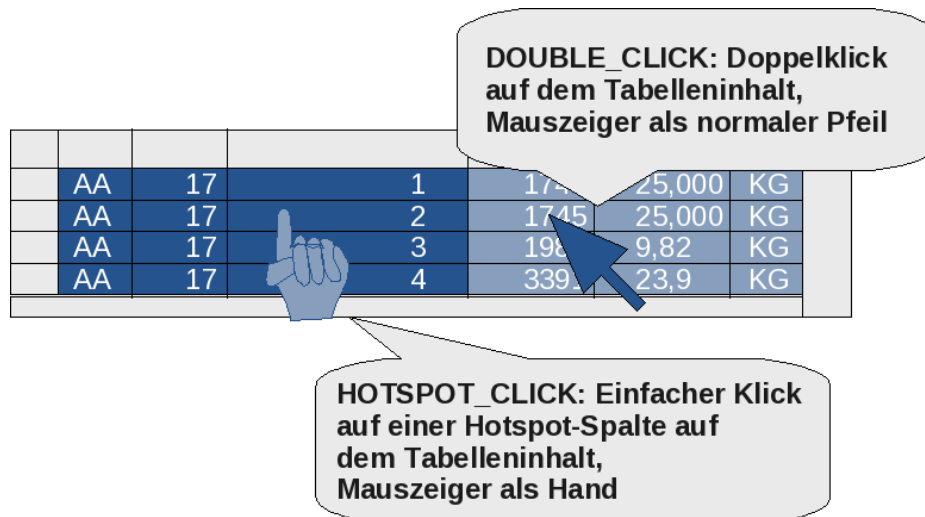


Abbildung 23: Klick-Ereignisse und Mauszeiger

Zur Behandlung der Ereignisse muss eine Klasse mit einer entsprechenden Behandlungsmethode geschrieben und die Methode auf die Ereignisse registriert werden.

8.6 Praxis: Übung zur Ereignisbehandlung

In dieser Übung soll das Programm ZZ_####_FCAT so erweitert werden, dass bei Doppelklick auf eine Buchung die Details des Kunden angezeigt werden.

Erstellen Sie hierzu im TOP-Include einen Arbeitsbereich zur SCUSTOM-Tabelle mit der TABLES-Anweisung. Öffnen Sie anschließend das Layout von Dynpro 300. Fügen Sie dort durch Übernahme aus dem Dictionary Felder für die Daten des Kunden ein. Stellen Sie sicher dass diese Felder nicht eingabebereit sind. Vergessen sie nicht das OK-Code-Feld zu benennen.

Abbildung 24: Dynpro 300 im Screen Painter: SAP-System-Screenshot

Für die Behandlung des Ereignisses, das durch den Doppelklick auf den ALV-Eintrag auf Dynpro 200 ausgelöst wird, benötigen Sie eine entsprechende Behandlermethode. Hierfür soll nun eine Klasse angelegt werden. Erzeugen Sie für diese Klasse ein neues Include **ZZ_####_FCAT_CLASSES**. Stellen Sie sicher, dass dieses im Rahmenprogramm **ZZ_####_FCAT** vor dem Include eingebunden wird, in dem das ALV Grid Control initialisiert wird.

Öffnen Sie das neue Include und erstellen Sie dort eine Klasse **lcl_event_handler**. Diese soll lediglich eine statische Methode **on_double_click** besitzen, die als Eventhandlermethode für das Ereignis **double_click** der Klasse **cl_gui_alv_grid** dienen und dessen Parameter **es_row_no** entgegennehmen soll. Machen Sie dies in der Definition entsprechend kenntlich.

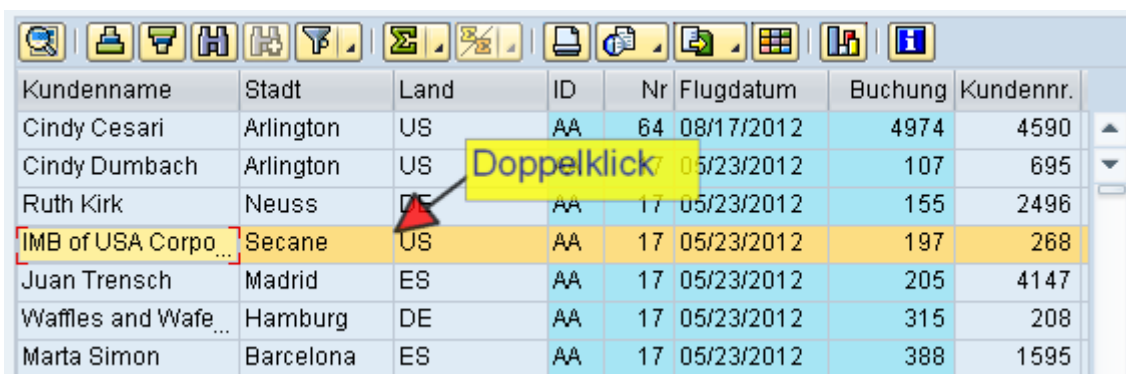
Implementieren Sie nun die Methode. Lesen Sie aus der internen Tabelle mit den Daten des ALV Grid über den **READ TABLE**-Befehl die angeklickte Zeile in einen passend typisierten Arbeitsbereich. Den dafür benötigten Zeilenindex erhalten Sie durch die Komponente **es_row_no-row_id**. Aus der gelesenen Zeile können Sie die Nummer des Kunden (**customid**) entnehmen, die Sie anschließend benutzen, um die Daten des Kunden mit einer SQL-Anfrage aus der Tabelle **SCUSTOM** einzulesen, so dass diese auf Dynpro 300 angezeigt werden. Sorgen Sie dafür, dass an dieser Stelle auch zu Dynpro 300 navigiert wird.

Damit das Ereignis behandelt wird, müssen Sie nun noch die Registrierung vornehmen. Öffnen Sie dazu das PBO-Modul von Dynpro 200, in dem das ALV Grid Control initialisiert wird. Ergänzen Sie dort nach der Erzeugung des ALV-Objektes die Registrierung:

```
SET HANDLER lcl_event_handler=>on_double_click FOR r_alv.
```

Stellen Sie zu guter Letzt noch sicher, dass von Dynpro 300 mit der Zurück-Schaltfläche wieder zu Dynpro 200 navigiert werden kann. Speichern, prüfen und aktivieren Sie alle zu aktivierenden Objekte. Bei der normalen Syntaxprüfung entsteht eine Fehlermeldung, da das Include in dem die Klasse definiert wird, nicht vom aktuellen Include eingebunden wird. Um stattdessen eine Prüfung durchzuführen, die alle Includes des Rahmenprogramms berücksichtigt, wählen Sie den Menüpfad **Programm -> Prüfen -> Rahmenprogramm**. Falls die Klasse dennoch nicht gefunden wird, prüfen Sie die Reihenfolge der Includes im Hauptprogramm: Das Include, das die Klasse enthält, muss vor dem Include eingebunden werden, in dem sie die Klasse verwenden. Außerdem ist es wichtig, dass das Hauptprogramm **aktiviert** ist!

Führen Sie dann das Programm über die Transaktion aus. Ein Doppelklick auf einen Eintrag führt nun zur Anzeige der Kundendaten.



The screenshot shows an SAP ALV Grid with columns: Kundenname, Stadt, Land, ID, Nr, Flugdatum, Buchung, and Kundennr. A red arrow points to the entry for 'IMB of USA Corp...' with a yellow callout box labeled 'Doppelklick'.

Kundenname	Stadt	Land	ID	Nr	Flugdatum	Buchung	Kundennr.
Cindy Cesari	Arlington	US	AA	64	08/17/2012	4974	4590
Cindy Dumbach	Arlington	US	AA	17	05/23/2012	107	695
Ruth Kirk	Neuss	DE	AA	17	05/23/2012	155	2496
IMB of USA Corp...	Secane	US	AA	17	05/23/2012	197	268
Juan Trench	Madrid	ES	AA	17	05/23/2012	205	4147
Waffles and Wafe...	Hamburg	DE	AA	17	05/23/2012	315	208
Marta Simon	Barcelona	ES	AA	17	05/23/2012	388	1595

Abbildung 25: Doppelklick auf einen Eintrag: SAP-System-Screenshot

Kundennummer	268
Kundenname	IMB of USA Corporation
Anrede	
Strasse	456 Bishop Ave
Postfach	
Postleitzahl	19018
Stadt	Secane
Land	US
Region	PA
Telefon	
<input type="checkbox"/> G/P-Kunde	
Rabatt(%)	0
Sprache	EN
E-Mail-Adresse	
Webname	

Abbildung 26: Anzeige der Details des gewählten Kunden: SAP-System-Screenshot

Sie haben nun die Behandlung von Ereignissen durch Benutzeraktionen an einem Beispiel praktisch angewendet. Im nächsten Abschnitt lernen Sie auch ein vom System automatisch ausgelöstes Ereignis kennen, das Sie zum Bearbeiten der Toolbar benutzen können.

8.7 Erweitern der Toolbar

Oberhalb des Tabelleninhalts eines ALV Grid Control befindet sich eine Toolbar mit verschiedenen Funktionen, die etwa die Sortierung der Tabelle oder die Filterung ermöglichen:

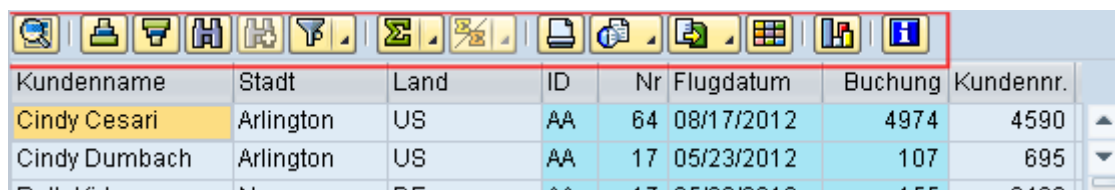


Abbildung 27: Toolbar eines ALV Grid Controls: SAP-System-Screenshot

Diese Toolbar kann um eigene Funktionen erweitert werden. Vor dem Einbinden der Toolbar in das ALV Grid Control (sowie bei Bedarf durch expliziten Aufruf der Methode **set_toolbar_interactive**) wird das Ereignis **toolbar** ausgelöst. Dieses besitzt einen Parameter **e_object** vom Typ (Referenz zu) **cl_alv_event_toolbar_set**. Darin findet sich wiederum das Attribut **mt_toolbar**. Hierbei handelt es sich um eine interne Tabelle, die die Einträge in der Toolbar enthält. Über diese Tabelle können benutzerdefinierte Funktionen zur Toolbar hinzugefügt werden.

Der Zeilentyp der internen Tabelle ist **stb_button**. Diese hat die folgenden Felder:

- **function:** Der zugeordnete Funktionscode
- **icon:** Der Name einer Ikone
- **quickinfo:** Die Quickinfo der Ikone
- **butn_type:** Typ des Elements. Mögliche Werte sind hier:

- 0 für einen gewöhnlichen Button
- 1 für ein Menü mit Button
- 2 für ein Menü
- 3 für einen Trenner
- 4 für ein Auswahlfeld
- 5 für ein Ankreuzfeld
- 6 für einen Menüeintrag
- **disabled:** Deaktiviert
- **text:** Text für den Button
- **checked:** Angabe ob der Button gedrückt sein soll

Für eine Benutzerdefinierte Funktion in der Toolbar muss ein neuer Eintrag dieser Form in die Tabelle eingetragen werden. Ein Eintrag vom Typ 1 oder 2 löst bei Auswahl das Ereignis **menu_button** mit dem Parameter **e_object** der Klasse **cl_ctmenu** aus, bei dessen Behandlung das Menü angelegt werden kann. Verschiedene Menüs werden über ihren Funktionscode unterschieden. Die Menüerstellung wird hier nicht näher betrachtet.

Klickt der Anwender auf einen der benutzerdefinierten Buttons in der Toolbar, wird das Ereignis **user_command** ausgelöst. Dieses besitzt einen Parameter **e_ucomm**, mit dem der Funktionscode übermittelt wird.

8.8 Praxis: Übung zum Erweitern der Toolbar

In dieser Übung werden Sie die Toolbar ihres ALV Grid Controls erweitern. Es soll ein zusätzlicher Button erscheinen, mit dem die Bonuspunkte des Kunden angezeigt werden können. Der Kunde erhält pro Flug einen Bonus, etwa pauschal 100 Bonuspunkte zuzüglich 1 Bonuspunkt pro 10 Minuten Flugzeit. Diese Bonuspunkte können später für Rabattaktionen eingesetzt werden.

Zunächst muss die Toolbar um den Button ergänzt werden. Öffnen Sie hierzu Ihre Klasse **lcl_event_handler** und ergänzen Sie im Definitionsteil der Klasse eine Methode **on_toolbar**, die Sie als Behandlungsmethode des Ereignisses **toolbar** der Klasse **cl_gui_alv_grid** mit dessen Parameter **e_object** deklarieren. Erstellen Sie anschließend die Implementierung der Methode. Erstellen Sie dort einen Arbeitsbereich **wa_button** vom Typ **stb_button**. Setzen Sie dann die Komponenten des Arbeitsbereichs wie folgt:

Wählen Sie als Funktionscode des Buttons 'BONUS', als Quickinfo „Bonuspunkte“, Als Typ 0 und als Text „Bonus“.

Fügen Sie den Button anschließend in die Toolbar ein, indem Sie ihn der internen Tabelle **e_object->mt_toolbar** hinzufügen. Sorgen Sie dann dafür, dass die Behandlungsmethode auf die entsprechenden Ereignisse Ihres ALV Grid registriert wird. Einen solchen SET HANDLER-Aufruf haben Sie bereits für das Doppelklick-Ereignis implementiert, ergänzen Sie den neuen Aufruf dort analog. Sichern und aktivieren Sie alle nicht aktivierten Objekte und starten Sie das Programm über die Transaktion. Der Button ist nun sichtbar:

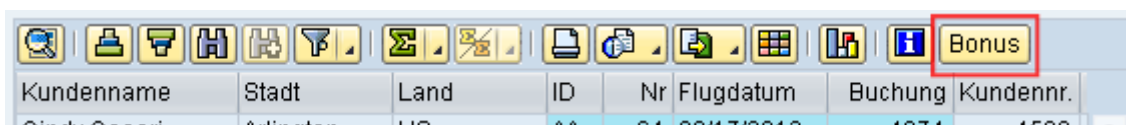


Abbildung 28: Neuer Button in der Toolbar: SAP-System-Screenshot

Es fehlt noch die Ereignisbehandlung beim Anklicken des Buttons. Öffnen Sie daher wieder Ihre Klasse und ergänzen Sie eine weitere (ebenfalls statische) Behandlungsmethode. Sie soll den Namen **on_user_command** besitzen, das Ereignis **user_command** der Klasse **cl_gui_alv_grid** behandeln und dessen Parameter **e_ucomm** entgegennehmen können.

Erstellen Sie als nächstes die Implementierung dieser Methode. Damit Sie den Bonus berechnen können, müssen Sie wissen, welche Zeile des ALV Grid der Benutzer gerade ausgewählt hat. Hierfür steht die Methode **get_current_cell** des ALV-Objekts zur Verfügung. Diese besitzt zwei Exportparameter: **e_row** liefert die Zeilennummer, **e_col** die Spaltennummer. Verwenden Sie die Zeilennummer, um mit dem **READ TABLE**-Befehl die entsprechende Zeile aus der internen Tabelle mit den Daten des ALV Grids auszulesen. Dieser können Sie die Fluggesellschaft und die Flugverbindung entnehmen. Rufen Sie mit diesen Informationen die Flugdauer aus der Datenbanktabelle **SPFLI** ab. Berechnen Sie den Bonus und geben Sie diesen mit einer geeigneten Nachricht (**MESSAGE ... TYPE 'I' .**) aus. *Hinweis: Um Zeichenketten zu verketteten, können Sie den **CONCATENATE**-Befehl verwenden. Um eine Zahl in eine Zeichenkette umzuwandeln, weisen Sie die Variable einer entsprechend Typisierten Variable zu.*

Ergänzen Sie auch für dieses Ereignis die entsprechende Registrierung der Behandlungsmethode. Speichern und aktivieren Sie alle Objekte und testen Sie das Programm über die Transaktion.

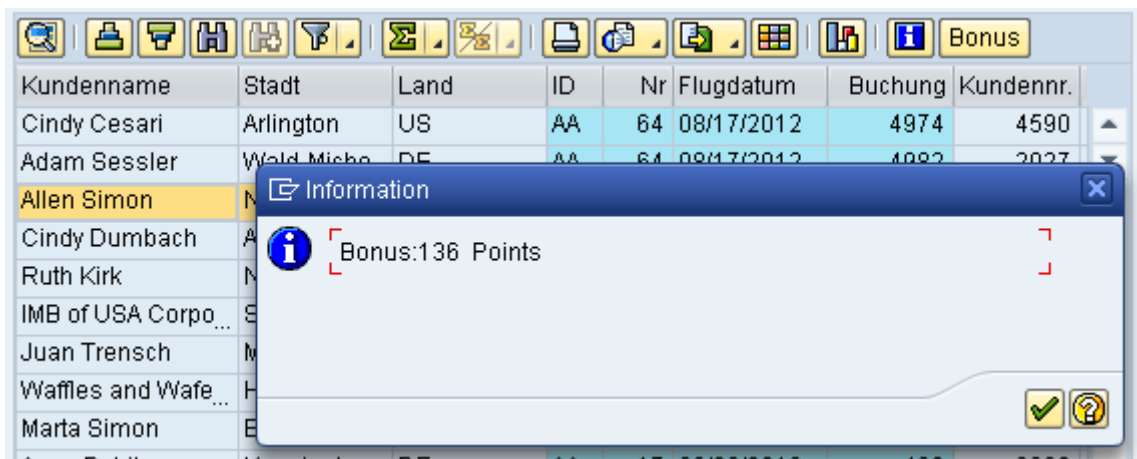



Abbildung 29: Ausgabe der Bonuspunkte: SAP-System-Screenshot

Sie haben nun die Toolbar um einen eigenen Button mit einer selbstdefinierten Funktionalität ergänzt. Dazu haben Sie die Ereignisse **toolbar** und **user_command** des ALV Grid benutzt. Weiterhin haben Sie die Methode **get_current_cell** verwendet, um die aktuelle Zeile zu bestimmen.

8.9 Weitere Ereignisse und Methoden des ALV Grid Control

Über die in der letzten Übung kennengelernte Methode zur Bestimmung der aktiven Zeile bzw. Spalte hinaus gibt es noch eine Vielzahl weiterer Methoden im ALV Grid Control. Auch gibt es weitere Ereignisse.

Das ALV Grid kann nicht nur im Dynpro angezeigt werden, sondern auch als gewöhnliche Listenausgabe. Wählen Sie dazu  (**Ansichten**) und **Listausgabe**:

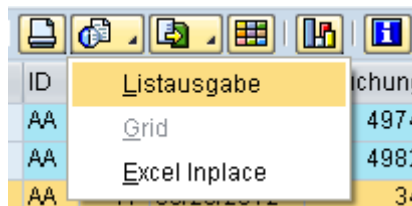


Abbildung 30: Aufruf der Listausgabe: SAP-System-Screenshot

Daraufhin erscheint die Ausgabe als Liste, wie Sie mit Befehlen wie WRITE, ULINE usw. erzeugt werden kann. Diese Liste wird automatisch von der repräsentativen Instanz generiert.

Buchungsprogramm

Kundenname	Stadt	Land	ID	Nr	Flugd
Cindy Cesari	Arlington	US	AA	64	08/17
Adam Sessler	Wald-Michelbach	DE	AA	64	08/17
Allen Simon	N. Massapequa	US	AA	17	05/23
Cindy Dumbach	Arlington	US	AA	17	05/23
Ruth Kirk	Neuss	DE	AA	17	05/23
IMB of USA Corporation	Secane	US	AA	17	05/23
Juan Trench	Madrid	ES	AA	17	05/23

Abbildung 31: Listausgabe des ALV Grid Control: SAP-System-Screenshot

Für diese Listenausgabe gibt es einige Ereignisse, die behandelt werden können, um das Aussehen der Liste, insbesondere für den Druck zu beeinflussen.

In der Regel ist für den Benutzer in seinen Benutzervorgaben ein Drucker eingerichtet, über den er die Ausgaben ausdrucken kann. Hierfür werden Spool-Aufträge erstellt. Die Erstellung der Spool-Aufträge ist nicht Aufgabe des Spool-Workprozesses. Sie kann im Dialog oder im Hintergrund von den jeweiligen Workprozessen erledigt werden. Hintergrund-Jobs bieten sich dann an, wenn die Verarbeitung sehr lange dauert (Dialogprogramme werden nach 10 Minuten abgebrochen) und besser zu Lastarmen Zeiten ausgeführt werden sollte. Die Jobs warten dann in der Jobeinplanungstabelle bis sie vom Background Scheduler auf einen Hintergrund-Workprozess verteilt werden. Eigene Jobs finden Sie im Menü unter **System-> Eigene Jobs** oder in der Transaktion **SMX**, sie können in der Transaktion **SM36** angelegt und in der Transaktion **SM37** systemweit überwacht werden.

Der Spool-Auftrag beinhaltet Informationen über die Daten, die ausgegeben werden sollen, sowie deren Aufbereitung und das Druckermodell. Der Spool-Workprozess übernimmt dann die Aufbereitung der Daten des Spool-Auftrags und erzeugt einen Ausgabeauftrag. Dieser enthält die Daten in einem für den Drucker geeigneten Format. Die Daten werden dann an den Betriebssystem-Spool-Prozess (**lokal** auf dem gleichen System oder **remote** per Netzwerk) übertragen. Neben lokal und remote gibt es weitere sog. Koppelarten, die den Zugriff auf den Betriebssystem-Spool-Prozess beschreiben. Die Koppelart lokal ist besonders performant. Das Betriebssystem ist dann für alle weiteren Aufgaben zuständig, etwa das Queuing und die Übermittlung der Daten an den Drucker. Eigene Spoolaufträge können mit der Transaktion **SP02** angezeigt werden, während die Transaktion **SU3** auf der Registerkarte **Festwerte** im Abschnitt **Spool-Steuerung** persönliche Voreinstellungen zum Druck ermöglicht.

Die folgende Abbildung zeigt die Ereignisse, die während der Listverarbeitung entstehen.

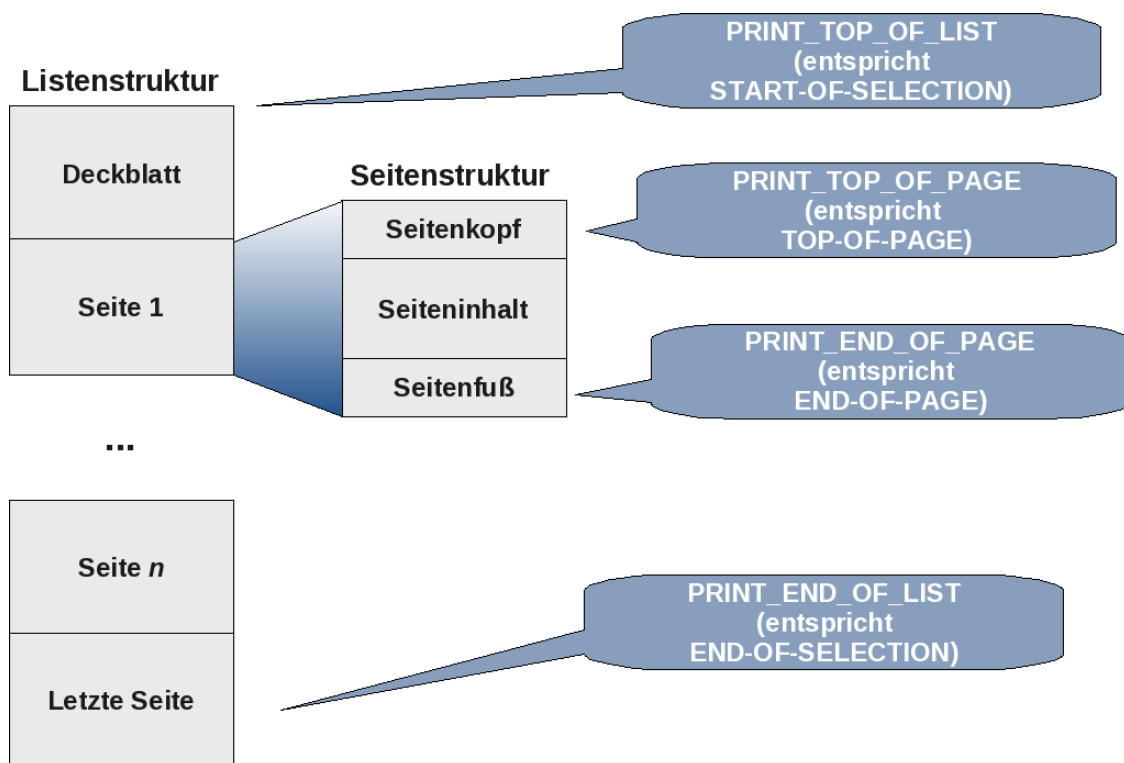


Abbildung 32: Ereignisse der Listenausgabe

Bei Behandlung der Ereignisse können Listenebefehle wie `WRITE` oder `ULINE` verwendet werden. Speziell für die Beschriftung von Zwischensummen kann das Ereignis **subtotal_text** verwendet werden.

Es ist auch möglich, Gruppen von Datensätzen durch Unterstreichungen oder Seitenwechsel in der Druckausgabe voneinander abzuheben. Hierfür muss zunächst der Parameter **is_print** der Methode **set_table_for_first_display** gesetzt werden. Hier wird eine Struktur vom Typ **lvc_s_pnrt** erwartet. Diese besitzt eine Komponente **grpchgedit** die auf 'X' gesetzt werden muss. Daraufhin hat der Anwender, der das Programm benutzt, die Möglichkeit beim Wechsel des Wertes von Sortierkriterien in der Listenausgabe eine Hervorhebung durch Unterstreichung oder Unterstreichung mit Seitenwechsel einzustellen. Dies wird in der folgenden Übung praktisch demonstriert. Ohne Benutzerbeteiligung ist eine Einstellung über den Parameter **it_sort** der Methode **set_table_for_first_display** und dessen Komponente **group** möglich.

Mit der an **is_print** übergebenen Struktur lassen sich außer Gruppenerhebungen noch weitere Einstellungen über die entsprechenden Komponenten vornehmen:

- **reserve_lns** zum Reservieren von Zeilen für den Fuß jeder Seite
- **print** zum Drucken ohne Bildschirmausgabe
- **prntlstinf** zur Ausgabe von Daten über die Sortierung, Filtereinstellungen usw. am Beginn der Liste
- **prnt_title** zur Ausgabe des Druckdatums im Titel
- **no_colwopt** zum Abschalten der Optimierung der Spaltenbreite

Weiterhin ist es möglich, die in der Toolbar vordefinierten Funktionen durch eigene Funktionen zu ersetzen. Hierzu muss das Ereignis **before_user_command** behandelt werden, das vom ALV Grid beim Anklicken einer Funktion der Toolbar ausgelöst wird. Dieses Ereignis besitzt einen Parameter **i_ucomm**, dem Sie den Funktionscode des angeklickten Buttons entnehmen können. In der Behandlermethode kann dann entsprechend reagiert

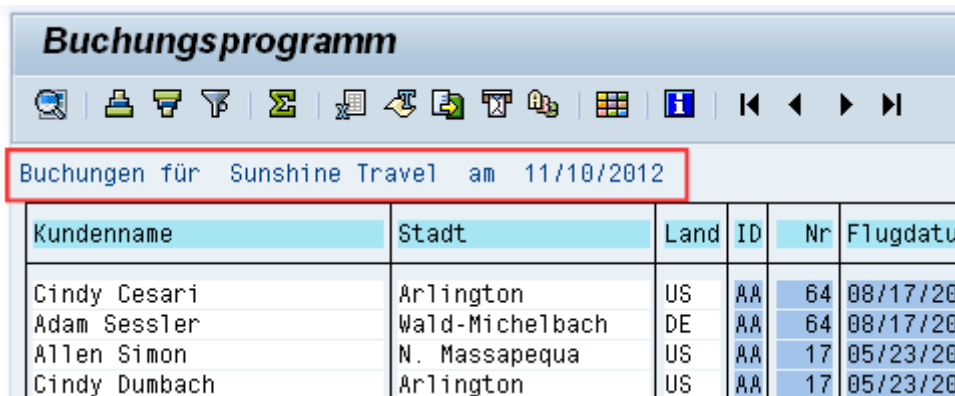
werden. Um zu verhindern, dass zusätzlich die ursprüngliche (von SAP definierte) Funktion des Buttons ausgeführt wird, setzen Sie den Funktionscode in der Behandlungsmethode über die Methode **set_user_command** zurück.

8.10 Praxis: Übung zur Formatierung der Druckliste

In dieser Übung werden Sie die Druckliste um einen einleitenden Satz erweitern und dem Benutzer das Hervorheben von Datensatzgruppen auf derselben ermöglichen.

Ergänzen Sie die Klasse **lcl_event_handler** um eine statische Methode **on_top_of_list**. Hierbei soll es sich um eine Ereignisbehandlungsmethode für das Ereignis **print_top_of_list** der Klasse **cl_gui_alv_grid** handeln. Kennzeichnen Sie die Methode entsprechend.

Erstellen Sie als nächstes die Implementierung der Methode. Die Einleitung der Liste soll den Stand (aktuelles Datum) und den Namen des Reisebüros enthalten. Den Reisebüronamen können Sie dem Feld **name** der Datenbanktabelle **stravelag** entnehmen (d.h. sie müssen hier zunächst eine **SELECT**-Anweisung implementieren, um den Namen aus der Datenbank auszulesen). Zur Ausgabe genügt eine gewöhnliche **WRITE**-Anweisung. Fügen Sie schließlich noch einen entsprechenden **SET HANDLER**-Aufruf zu den anderen Behandler-Registrierungen hinzu. Speichern und aktivieren Sie alle entsprechenden Objekte und testen Sie das Programm. Die Listausgabe (also nicht das Dynpro, sondern die Ausgabe nach Auswahl von Ansichten -> Listausgabe) sollte nun ihren Einleitungssatz enthalten:




Kundenname	Stadt	Land	ID	Nr	Flugdatum
Cindy Cesari	Arlington	US	AA	64	08/17/20
Adam Sessler	Wald-Michelbach	DE	AA	64	08/17/20
Allen Simon	N. Massapequa	US	AA	17	05/23/20
Cindy Dumbach	Arlington	US	AA	17	05/23/20

Abbildung 33: Ausgabe von Reisebüroname und Datum: SAP-System-Screenshot

Als nächstes soll die Gruppierung von Einträgen durch den Benutzer wie zuvor erläutert ermöglicht werden.

Öffnen Sie dazu das PBO-Modul, in dem die Methode **set_table_for_first_display** aufgerufen wird. Definieren Sie vor dem Methodenaufwurf eine Variable **my_print** vom Typ **lvc_s_prnt**. Setzen Sie deren Komponente **grpchgedit** auf 'X' und übergeben Sie die Variable an den Parameter **is_print** der Methode. Vergessen Sie nicht, den Kommentarstern vor dem Parameter zu entfernen. Sichern Sie das Include, aktivieren Sie und starten Sie das Programm.

Wählen Sie ein Reisebüro und wechseln Sie zur **Listausgabe** (bleiben Sie also nicht dem Dynpro, sondern wählen Sie zunächst den Ansichten-Button und **Listausgabe**). Klicken Sie dort auf  (**Sortieren Aufsteig**). Es erscheint ein Fenster, in dem Sie die Spalten auswählen können, nach denen sortiert werden soll. Wählen Sie aus dem Feldvorrat auf der rechten Seite

den Eintrag Land und klicken Sie auf  (**selektierte Felder einblenden**). Der Eintrag wechselt so auf die linke Seite zu den Sortierfeldern (siehe folgende Abbildung).

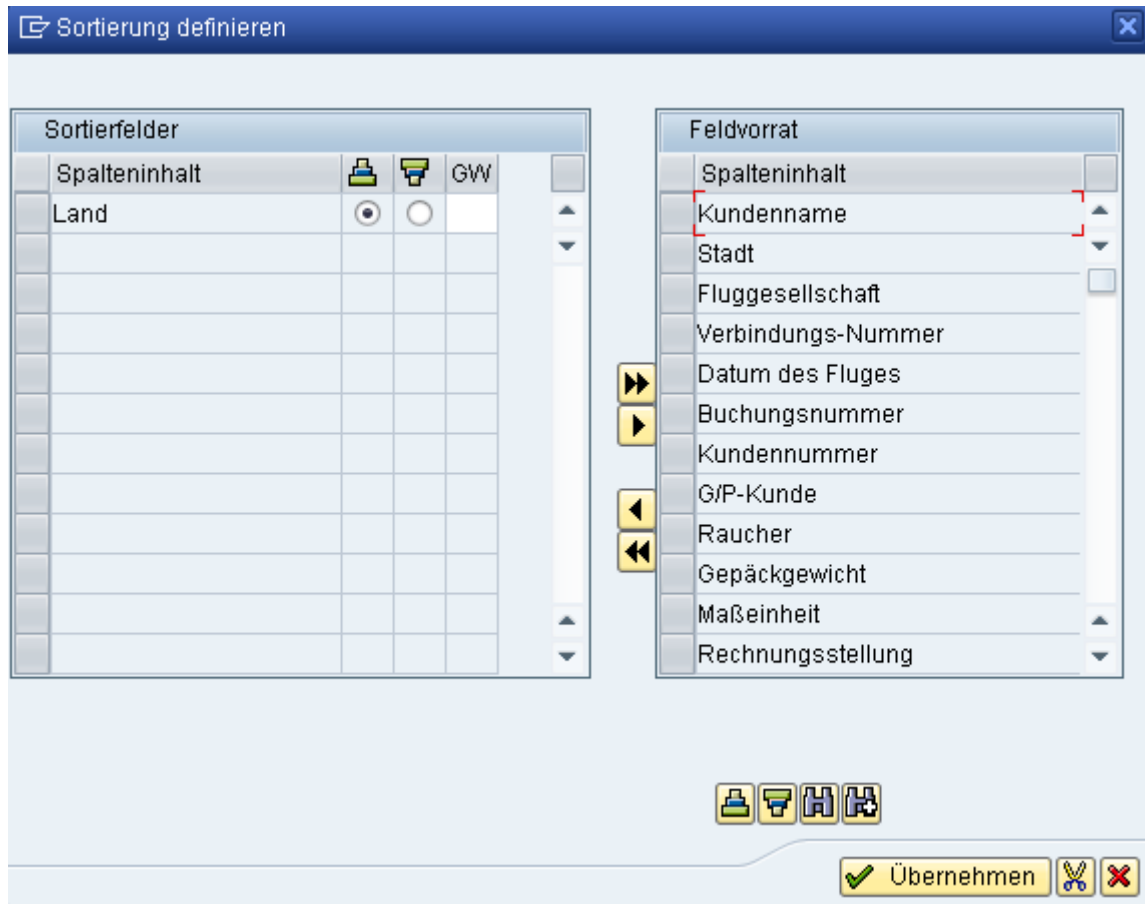


Abbildung 34: Sortierung definieren: SAP-System-Screenshot

Sie sehen dort auch eine Spalte **GW**. Diese Spalte ist durch die von Ihnen im Code vorgenommene Einstellung hinzugekommen. GW steht hier für **Gruppenstufenwechsel**. Nutzen Sie die F4-Hilfe um einen Wert auszuwählen, der zur Unterstreichung führt, und übernehmen Sie die Einstellungen.

Die Liste ist daraufhin nach Land sortiert. Zusätzlich sind die Einträge mit demselben Land durch einen Strich von den jeweils nächsten Einträgen getrennt:

Kundenname	Stadt	Land	ID
Jean Bentink	Bruxelles	BE	DL
Jean Detemple	Bruxelles	BE	AA
D. & P. Forbes	Toronto, Ontario	CA	AZ
IDES Canada	Toronto	CA	DL
IDES Canada	Toronto	CA	SQ
Simon Henry	Genève	CH	AZ
Simon Sommer	Genève	CH	AZ

Abbildung 35: Gruppierte Einträge: SAP-System-Screenshot

8.11 Kontrollfragen

Welche der folgenden Aussagen sind wahr/falsch?

1. Das ALV Grid Control-Objekt muss vor dem Objekt für das Container-Control erzeugt werden.
2. Es gibt Möglichkeiten zur Übernahme von Spaltenbeschreibungen aus dem ABAP Dictionary.
3. Für ein ALV Grid Control muss der Entwickler einen Feldkatalog aufbauen und für alle Felder dort Einträge vornehmen.
4. Die vordefinierten Einträge in der Symbolleiste des ALV Grid Control können nicht verändert werden.
5. Die repräsentative Instanz eines ALV Grid Controls befindet sich auf dem Anwendungsserver.
6. Änderungen an der Oberfläche (z. B. Sortierung) wirken sich auf die Datentabelle des ALV Grid Controls aus.

8.12 Antworten

1. Falsch (Reihenfolge ist umgekehrt)
2. Richtig
3. Falsch (automatische Generierung ist möglich)
4. Falsch (sie können durch eigene Funktionen ersetzt werden)
5. Richtig
6. Richtig

9 ABAP und Unicode

„There ain't no such thing as plain text” – dieser Satz aus einem Artikel über Unicode von Joel Spolsky macht darauf aufmerksam, dass jeder Text, der in einem Softwaresystem verarbeitet werden soll, eine Codierung benötigt. Viele Programmierer sind sich dessen nicht bewusst, was etwa zu Webseiten oder E-Mails mit falsch dargestellten Umlauten oder anderen Sonderzeichen führt.

In einem Softwaresystem müssen die Zeichen als Bitsequenzen repräsentiert werden. Je nachdem wie viele Bit für ein Zeichen vorgesehen sind unterscheidet sich die Anzahl der unterscheidbaren Zeichen in diesem System. Beim ASCII-Zeichensatz sind für jedes Zeichen 8 Bit (1 Byte) vorgesehen, so dass 256 Zeichen unterschieden werden können. Sobald andere Zeichen als die im Zeichensatz vorgesehenen benötigt werden, ist das Laden eines anderen Zeichensatzes erforderlich, der eine andere Zuordnung zwischen den Bits und Zeichen definiert. Der Austausch von Daten zwischen Benutzern mit verschiedenen Zeichensätzen wird dadurch erschwert. Um hier Abhilfe zu schaffen, wurde **Unicode** erfunden. Durch die großzügige Bemessung von 16 Bit wurden so 65536 verschiedene Zeichen möglich – genug, um alle bisherigen Zeichensätze durch Unicode ersetzen zu können. Seit Release 6.10 bietet auch das SAP-System Unicode-Unterstützung.

ABAP bietet mit C, N, D, T und STRING eine ganze Reihe von zeichenbasierten Datentypen. Auf Systemen, die nicht mit Unicode arbeiten, wird jedes Zeichen durch ein Byte repräsentiert. Auf Unicode-Systemen gilt dies nicht – hier beansprucht jedes Zeichen die Länge eines Zeichens auf der jeweiligen Plattform. Ein Programm das mit Unicode arbeitet, sollte dasselbe Verhalten auch auf Nicht-Unicode-Systemen aufweisen. In den Eigenschaften jedes Programms befindet sich ein Häkchen **Unicodeprüfungen aktiv**, mit dem erweiterte Syntaxprüfungen eingeschaltet werden, die ebendies sicherstellen. Ohne Setzen dieses Häkchens kann das Programm nur auf einem Nicht-Unicode-System ausgeführt werden.

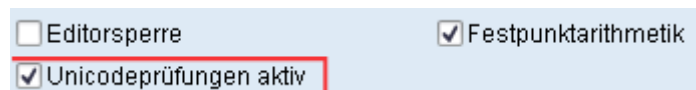


Abbildung 36: Aktivieren der Unicodeprüfungen: SAP-System-Screenshot

Um die Kompatibilität sicherzustellen, erwarten Befehle auf Zeichenketten in ihrer normalen Form stets eine Zeichenkette und verarbeiten diese zeichenweise. Es ist aber auch möglich auf Byte-Ebene zu arbeiten. Dies ist für die Typen X und XSTRING sinnvoll, die Byte-Ketten sind. Ein Umschalten zwischen beiden Befehlsvarianten ist über den Zusatz IN BYTE MODE bzw. IN CHARACTER MODE möglich.

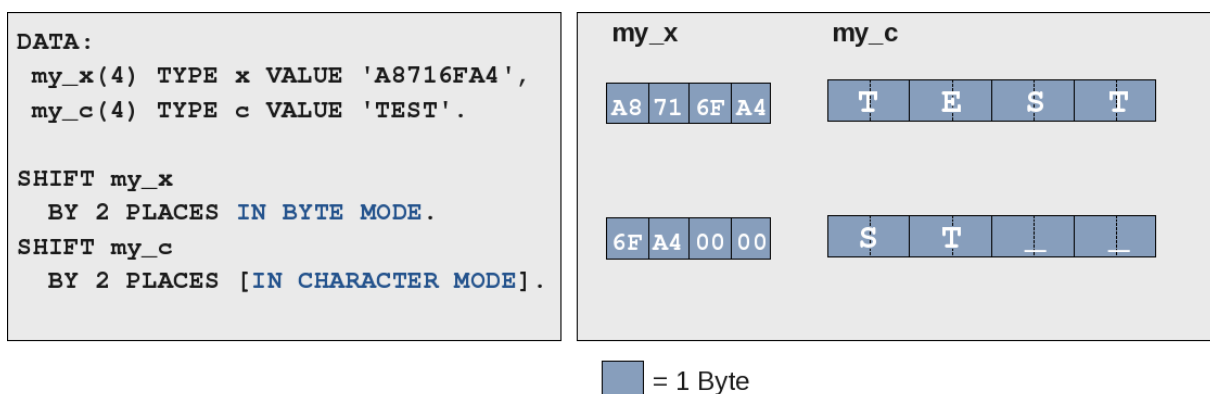


Abbildung 37: Byte- und Character-Mode

Die obige Abbildung zeigt ein Beispiel. Die Variablen `my_x` und `my_c` wurden beide mit der Länge 4 definiert. Intern werden die Zeichen als zwei Bytes repräsentiert. Der `SHIFT`-Befehl, der den Inhalt einer Variablen verschiebt, verhält sich dem Zusatz entsprechend unterschiedlich bei der Byte- und der Zeichenkette.

Aufgrund dieser unterschiedlichen internen und externen Repräsentation sind auch eigene Funktionen und Operatoren sinnvoll, die die jeweilige Repräsentation berücksichtigen. Eine Ermittlung der Länge ist z. B. klassisch über den `DESCRIBE FIELD`-Befehl möglich, der allerdings auch zwischen `BYTE`- und `CHARACTER MODE` unterscheidet und bei Datenobjekten des Typs `String` stets 8 zurückliefert. Um diese Missstände zu beheben wurden die folgenden Funktionen und Operatoren eingeführt:

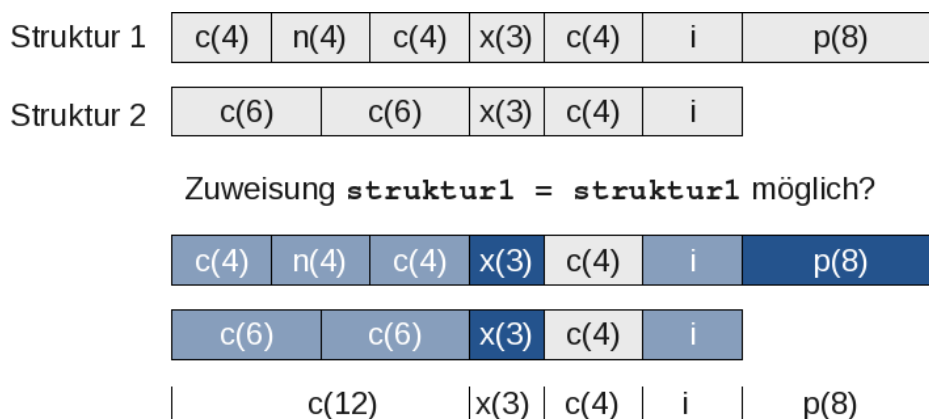
Tabelle 5: Funktionen und Operatoren für Zeichen- bzw. Bytebasierte Typen

	Zeichenbasierter Typ	Bytebasierter Typ
Funktionen	<code>STRLEN ()</code>	<code>XSTRLEN ()</code>
Operatoren	<code>CO, CA, CS, CP, CN, NA, NS, NP</code>	<code>BYTE-CO, BYTE-CA, BYTE-CS, BYTE-CN, BYTE-NA, BYTE-NS</code>

Während `STRLEN` die Länge als Anzahl von Zeichen bestimmt, berechnet `XSTRLEN` die Länge in Bytes. `STRLEN` zählt bei Datenobjekten des Typs `c` etwaige Leerzeichen am Ende nicht mit. `XSTRLEN` liefert beim Typ `x` die definierte Länge, beim Typ `XSTRING` die aktuelle Länge. Für nähere Informationen zu den einzelnen Operatoren verwenden Sie die Schlüsselwortedokumentation.

Es ist auch möglich, Strukturen unterschiedlichen Typs einander zuzuweisen (was wir bislang nicht getan haben). Das System betrachtet dabei die Fragmente der jeweiligen Struktur aus Bereichen von Byte-Typen, Bereichen von Zeichen-Typen, Alignment Gaps (Plattformabhängige Füllbytes um bestimmte Speicheradressen zu erreichen) und sonstige Typen und prüft, ob eine Zuweisung möglich ist.

Die Zuweisung ist möglich, wenn die Fragmente von Typ und Länge kompatibel sind. Ist die Zielstruktur länger als die Quellstruktur werden die übrigen Zeichenbasierten Komponenten der Zielstruktur mit Leerzeichen gefüllt, während alle anderen verbleibenden Komponenten den typgerechten Initialwert erhalten.



Fragmente kompatibel – Zuweisung möglich!

Abbildung 38: Fragmente von Strukturen

Nach diesem Prinzip ist es auch möglich, eine Struktur einem elementaren Datenobjekt vom Typ `c` zuzuweisen. Die Struktur muss entweder nur aus Komponenten von Zeichen-Typen bestehen oder mit einem Fragment von Komponenten von Zeichen-Typen beginnen, das mindestens so lang ist wie das elementare Datenobjekt. Solche Zuweisungen sind häufig schlecht lesbar und sollten daher vermieden werden. Vorzuziehen ist stattdessen eine Zuweisung der einzelnen Komponenten.

Auch beim Zugriff mit Offset und Länge unterscheiden sich Zeichen-Typen und Byte-Typen. Ein solcher Zugriff ist mit folgender Syntax möglich:

```
befehl feld+offset(laenge).
```

Hier wird dem Befehl `befehl` ein Ausschnitt aus dem Feld `feld` übergeben, der bei `offset` beginnt und die Länge `laenge` besitzt. Die Angabe von `offset` und `laenge` bezieht sich bei Zeichen-Typen auf die Zeichen, bei Byte-Typen hingegen auf die Bytes.

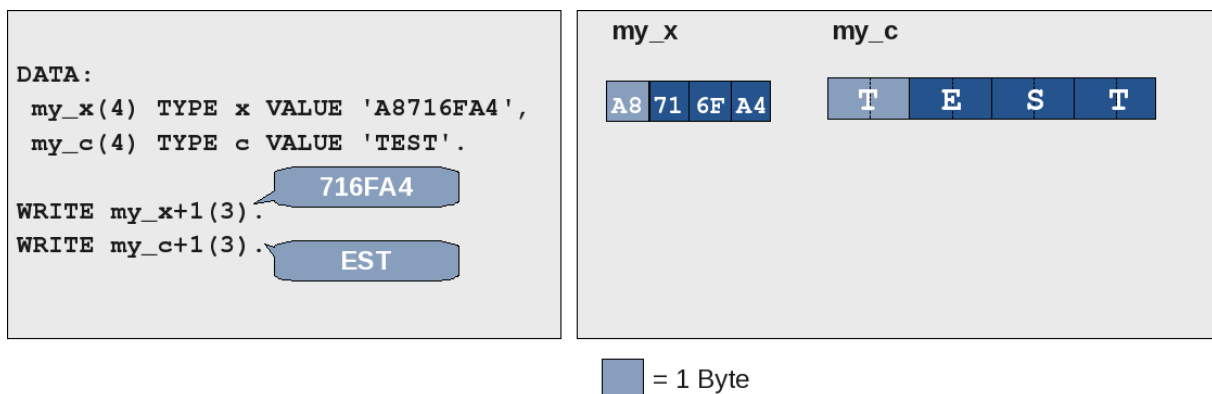


Abbildung 39: Zugriff mit Offset und Länge

Auch auf Strukturen kann auf diese Weise zugegriffen werden. Dies ist in Unicode-Programmen nur dann möglich, wenn die Struktur flach ist und mit einem Zeichen-Fragment beginnt und sich der Zugriff auf dieses Fragment beschränkt. Es gilt hier derselbe Hinweis wie oben: Es sollte gut überlegt sein, solche Zugriffe zu verwenden.

Insgesamt betrachtet ist es wichtig sicherzustellen, dass das Verhalten der Programme unabhängig von der Codierung ist. Das Setzen des Häkchens für die Unicodeprüfung sorgt dafür, dass entsprechende Syntaxwarnungen ausgegeben werden und das Programm gemäß den Unicode-Regeln ausgeführt wird. Programme, bei denen die Unicodeprüfungen nicht aktiv sind, können nur auf Nicht-Unicode-Systemen ausgeführt werden. Auf diesen Systemen wird durch den Profilparameter **abap/unicode_check** festgelegt, ob Unicodeprüfungen verwendet werden sollen. Hat dieser Parameter den Wert **on**, werden die Prüfungen für alle Programme (unabhängig von deren Einstellung) vorgenommen. Beim Wert **off** werden die Prüfungen nur bei Programmen durchgeführt, bei denen sie explizit aktiviert sind. Die Möglichkeit der globalen Aktivierung durch Wählen von **on** wird meist dann verwendet, wenn das System zu einem Unicode-System umgewandelt werden soll. Wenn in diesem Zuge mehrere Programme auf einen Schlag auf Unicode-Kompatibilität geprüft werden sollen, hilft die Transaktion **UCCHECK**. Mit dieser Transaktion können auch auf einer Menge von Programmen die Unicodeprüfungen aktiviert werden, solange diese im aktuellen System original sind.

10 Dynamische Programmierung

Mit den Mitteln der dynamischen Programmierung ist es möglich, sehr flexible Programme zu schreiben. ABAP bietet hierbei eine großzügige Unterstützung an. Aufgrund des Charakters von ABAP als „gewachsene Sprache“ sind die syntaktischen Strukturen nicht immer ganz einheitlich. Dafür sind die dynamischen Programmierkonzepte aber gut in die ABAP Workbench integriert.

10.1 Feldsymbole

Wenn Sie schon mit vielen anderen Programmiersprachen gearbeitet haben, ist Ihnen möglicherweise das Konzept des Zeigers bekannt. ABAP bietet in diesem Zusammenhang **Feldsymbole** an. Hierbei handelt es sich um **dereferenzierte Zeiger**. Über ein Feldsymbol kann auf ein Datenobjekt zugegriffen werden, auf das gezeigt wird. Es wird also bei der Verwendung des Feldsymbols nicht der Zeiger selbst, sondern das Datenobjekt auf das gezeigt wird erreicht (es wird „durchgegriffen“). Dieses Verhalten wird auch als **Wertesemantik der Feldsymbole** bezeichnet.

Die Definition eines Feldsymbols erfolgt über den Befehl `FIELD-SYMBOLS`.

```
FIELD-SYMBOLS <feldsymbol> typisierung.
```

`feldsymbol` ist hier der Name des Feldsymbols. Die spitzen Klammern um den Namen gehören mit zur Syntax. Die Typisierung kann auf gewohnte Weise vorgenommen werden. Das Feldsymbol kann dann Datenobjekte des angegebenen Typs referenzieren. Darüber hinaus ist als Typisierung der generische Typ `TYPE ANY` möglich. So typisierte Feldsymbole können beliebige Datenobjekte referenzieren.

Um ein Datenobjekt zu einem Feldsymbol zuzuweisen, wird der `ASSIGN`-Befehl benutzt:

```
ASSIGN datenobjekt TO <feldsymbol>.
```

Wenn das Feldsymbol generisch typisiert wurde (d. h. über `TYPE ANY`), übernimmt es hierbei den Typ des Datenobjekts.

Die Zuweisung kann mit dem `UNASSIGN`-Befehl wieder zurückgenommen werden:

```
UNASSIGN <feldsymbol>.
```

Um zu prüfen, ob ein Feldsymbol gerade zugeordnet ist, d. h. auf einen Speicherbereich zeigt, kann in logischen Ausdrücken über `<feldsymbol> IS ASSIGNED` geprüft werden. Die folgende Abbildung zeigt beispielhaft die Verwendung der Befehle:

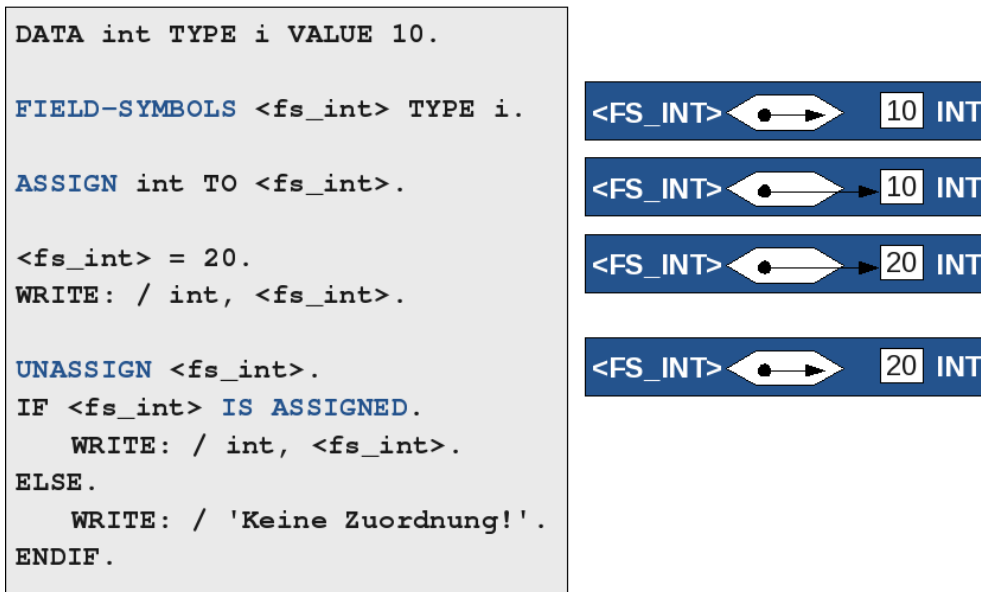


Abbildung 40: Anwendungsbeispiel für Feldsymbole

Zum Zeitpunkt der Zuweisung `<fs_int> = 20.` zeigt das Feldsymbol auf die Speicherstelle, in der zuvor die 10 gespeichert war. Diese wird überschrieben. Bei der folgenden Ausgabe würde also 20 20 angezeigt. Die Bedingung mit `IS ASSIGNED` ist negativ, da zuvor `UNASSIGN` aufgerufen wurde. Dadurch wird anschließend „Keine Zuordnung“ ausgegeben.

Bei der Zuweisung von Feldsymbolen kommt die Typisierung zum Tragen. Die generische Typisierung kann bewirken, dass eine Typinkompatibilität erst zur Laufzeit festgestellt werden kann, während statische Typinkompatibilitäten bereits bei der Prüfung gefunden werden. Das folgende Beispiel zeigt den Unterschied:

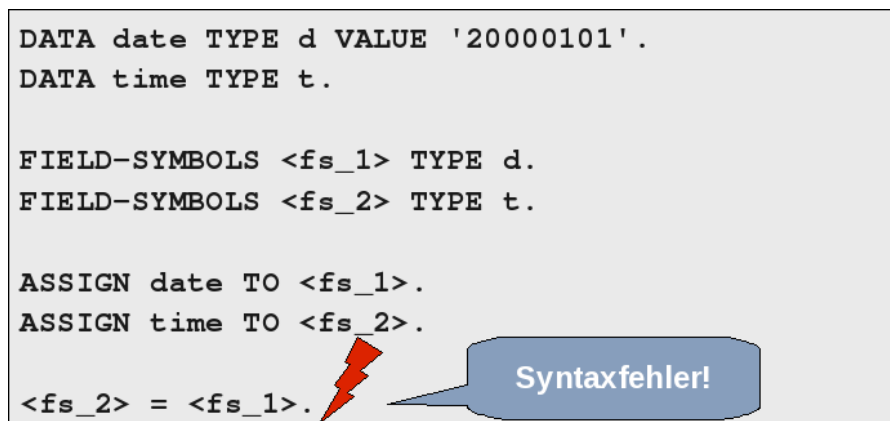


Abbildung 41: Code mit Syntaxfehler

Der hier abgebildete Code führt bei der Programmprüfung zu einem Fehler, da der Typ `d` nicht in den Typ `t` konvertierbar ist:

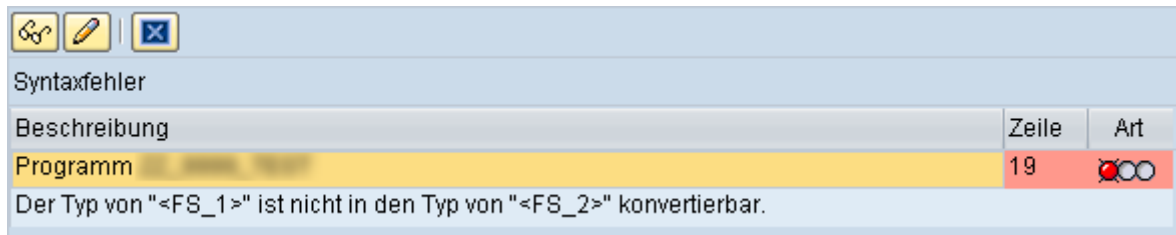


Abbildung 42: Fehlermeldung bei der Syntaxprüfung: SAP-System-Screenshot

Ein Ersetzen der Typisierung der Feldsymbole durch den generischen Typ ANY führt zu folgendem Code:

```
DATA date TYPE d VALUE '20000101'.
DATA time TYPE t.

FIELD-SYMBOLS <fs_1> TYPE ANY.
FIELD-SYMBOLS <fs_2> TYPE ANY.

ASSIGN date TO <fs_1>.
ASSIGN time TO <fs_2>.

<fs_2> = <fs_1>.
```

A red lightning bolt icon points to the last line of code, and a speech bubble next to it says "Laufzeitfehler!" (Runtime error!).

Abbildung 43: Code, der zu einem Laufzeitfehler führt

Dieser Code ist ebenso inkorrekt wie der zuvor gezeigte. Hier kann der Fehler aber nicht statisch (bei der Programmprüfung) festgestellt werden, sondern es wird während des Ausführens des Programms ein Laufzeitfehler ausgelöst:

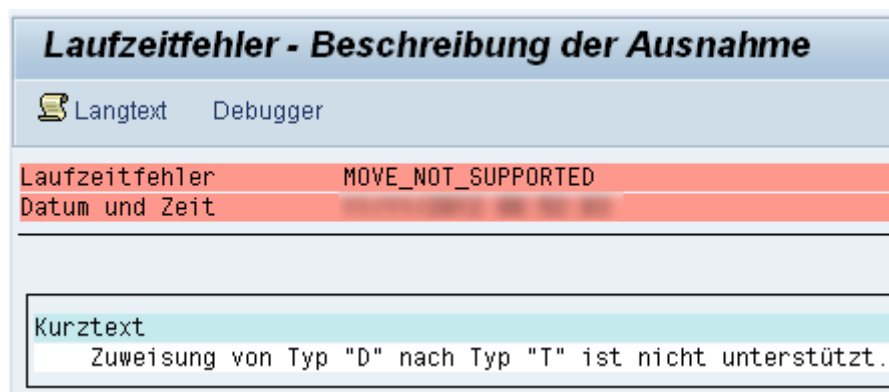


Abbildung 44: Laufzeitfehler: SAP-System-Screenshot

Einschränkungen des Typs bei der Zuweisung eines Datenobjekts an ein Feldsymbol können mit dem CASTING-Zusatz wieder aufgehoben werden. Der referenzierte Speicherbereich wird dann so behandelt, als hätte er den gewünschten Typ. Die folgende Abbildung zeigt ein Beispiel:

```

TYPES: BEGIN OF st_date,
        year(4)  TYPE n,
        month(2) TYPE n,
        day(2)   TYPE n,
        END OF st_date.

FIELD-SYMBOLS <fs> TYPE st_date.
ASSIGN sy-datum TO <fs> CASTING.
WRITE: / <fs>-year, <fs>-month, <fs>-day.

```

Abbildung 45: Der CASTING-Zusatz

Hier ist festzuhalten, dass `sy-datum` eine zeichenartige Komponente aus 8 Stellen ist. Nach der Zuweisung kann jedoch über die Komponenten der selbsterstellten Strukturdefinition zugegriffen werden, wie hier in der letzten Zeile.

Auch ein generisches Feldsymbol kann gecastet werden. In diesem Fall ist das Vorgehen aber komplizierter, da der Typ explizit angegeben werden muss. Weiterhin müssen die einzelnen Komponenten der Struktur zugreifbar gemacht werden, da diese sonst aufgrund der Typisierung syntaktisch nicht ansprechbar wären.

```

TYPES: BEGIN OF st_date,
        year(4)  TYPE n,
        month(2) TYPE n,
        day(2)   TYPE n,
        END OF st_date.

FIELD-SYMBOLS: <fs> TYPE ANY,
        <fs_year> TYPE st_date-year,
        <fs_month> TYPE st_date-month,
        <fs_day> TYPE st_date-day.
ASSIGN sy-datum TO <fs> CASTING TYPE st_date.
ASSIGN COMPONENT 1 OF STRUCTURE <fs> TO <fs_year>.
ASSIGN COMPONENT 2 OF STRUCTURE <fs> TO <fs_month>.
ASSIGN COMPONENT 3 OF STRUCTURE <fs> TO <fs_day>.
WRITE: / <fs_year>, <fs_month>, <fs_day>.

```

Abbildung 46: CASTING bei generischer Typisierung

Die Typangabe bei `CASTING TYPE` kann auch dynamisch vorgenommen werden, indem ein eingeklammertes Datenobjekt angegeben wird, welches zur Laufzeit den Namen des Typs enthält:

```
ASSIGN dobj TO <feldsymbol> CASTING TYPE (typname).
```

Beachten Sie, dass der Typname in Großbuchstaben angegeben werden muss. Die folgende Abbildung zeigt das Beispiel der vorangegangenen Abbildung unter Verwendung der dynamischen Typangabe:


```

TYPES: BEGIN OF st_date,
        year(4)  TYPE n,
        month(2) TYPE n,
        day(2)   TYPE n,
      END OF st_date.

DATA typename TYPE string VALUE 'ST_DATE'.

FIELD-SYMBOLS: <fs> TYPE ANY,
               <fs_year> TYPE ANY,
               <fs_month> TYPE ANY,
               <fs_day> TYPE ANY.
ASSIGN sy-datum TO <fs> CASTING TYPE (typename).
ASSIGN COMPONENT 1 OF STRUCTURE <fs> TO <fs_year>.
ASSIGN COMPONENT 2 OF STRUCTURE <fs> TO <fs_month>.
ASSIGN COMPONENT 3 OF STRUCTURE <fs> TO <fs_day>.
WRITE: / <fs_year>, <fs_month>, <fs_day>.

```

Großschreibung

Abbildung 47: Dynamische Typangabe beim Casting

Auch im objektorientierten Kontext stehen Feldsymbole zur Verfügung. So können sowohl statische als auch Instanzattribute referenziert werden:

```

ASSIGN class=>static_attribute TO <fs>.
ASSIGN r_object->instance_attribute TO <fs>.

```

Auch hier sind dynamische Angaben möglich. Die Angabe des Klassennamens und des Attributnamens können über entsprechende zeichenartige Datenobjekte vorgenommen werden:

```

ASSIGN (classname)=>(static_attribute_name) TO <fs>.
ASSIGN r_object->(instance_attribute_name) TO <fs>.

```

10.2 Datenreferenzen

Mit Release 4.6A kamen zum SAP-System zusätzlich zu den Feldsymbolen auch Datenreferenzen hinzu. Sie ermöglichen Referenzen auf Datenobjekte, wobei eine Referenzsemantik gilt:

Durch den Namen der Datenreferenz wird nicht das Datenobjekt angesprochen, sondern die Referenz selbst. Ein Zugriff auf das referenzierte Datenobjekt erfordert eine Dereferenzierung.

Die Definition einer Datenreferenz unterscheidet sich kaum von der einer Referenzvariablen auf ein Objekt. Auch hier wird bei der Typisierung die Syntax `TYPE REF TO` verwendet:

```

TYPES reference_type { TYPE REF TO type_name
                      | LIKE REF TO do_name
                      | TYPE REF TO data   }.

```

Analog kann direkt in der DATA-Anweisung typisiert werden:

```

DATA reference { TYPE REF TO type_name
                | LIKE REF TO do_name

```

```
| TYPE REF TO data  }.
```

Die Angabe `TYPE REF TO data` definiert eine generische Datenreferenz. Das Wort `data` ist ein hierfür reservierter Bezeichner (vergleichbar mit `constructor`). Um nun eine Referenz auf ein Datenobjekt zu erhalten, wird der Befehl `GET REFERENCE` verwendet:

```
GET REFERENCE OF dobj INTO reference.
```

Durch diesen Befehl wird in der Datenreferenz `reference` eine Referenz auf das Datenobjekt `dobj` gespeichert. Es wurde eingangs erwähnt, dass für einen Zugriff auf das Datenobjekt über die Datenreferenz eine Dereferenzierung erforderlich ist. Hierfür wird der Dereferenzierungs-Operator `->*` verwendet. Die folgende Abbildung zeigt ein Beispiel zur Verwendung von `GET REFERENCE` und diesem Operator:

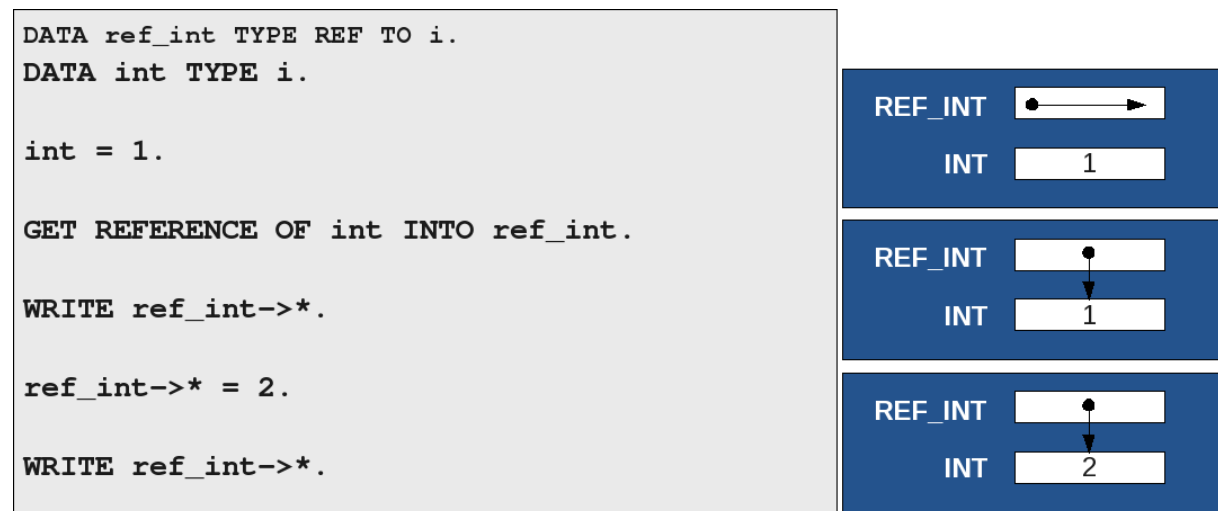


Abbildung 48: Verwendung von `GET REFERENCE` und Dereferenzierungs-Operator

Ohne den Dereferenzierungs-Operator würde auf die Referenz selbst zugegriffen. Diese enthält aber nur die Adresse des referenzierten Datenobjekts, nicht dessen Inhalt. Datenreferenzen verhalten sich selbst wie ein gewöhnliches Datenobjekt mit elementarem Datentyp, können also bspw. auch Teil von Strukturen sein.

Bei generisch typisierten Datenreferenzen ist ein direkter Zugriff auf das Datenobjekt über den Dereferenzierungs-Operator nicht möglich. Stattdessen muss die dereferenzierte Datenreferenz einem Feldsymbol zugewiesen werden, über das dann auf den Inhalt zugegriffen werden kann. Die folgende Abbildung zeigt ein Beispiel:

```
DATA ref_int TYPE REF TO data.  
DATA int TYPE i.  
  
int = 1.  
  
GET REFERENCE OF int INTO ref_int.  
FIELD-SYMBOLS <fs> TYPE i.  
  
ASSIGN ref_int->* TO <fs>.  
  
WRITE <fs>.
```

Abbildung 49: Dereferenzierung einer generischen Datenreferenz

Bei erfolgreicher Zuweisung wird durch die ASSIGN-Anweisung der sy-subrc auf 0 gesetzt. Ist die Datenreferenz initial bzw. ungültig, kann keine Dereferenzierung durchgeführt werden und sy-subrc erhält den Wert 4.

Bei Datenreferenzen auf Strukturen können die Strukturkomponenten über den Komponentenselektor -> angesprochen werden. Die folgende Abbildung zeigt ein Beispiel:

```
DATA: wa TYPE SPFLI,  
      ref TYPE REF TO SPFLI.  
  
GET REFERENCE OF wa INTO ref.  
  
ref->carrid = 'LH'.  
  
WRITE ref->carrid.
```

Abbildung 50: Zugriff auf Strukturkomponenten per Datenreferenz

Um zu prüfen, ob eine Referenz gültig ist, kann in einem logischen Ausdruck eine Bedingung der Form

```
... ref IS [NOT] BOUND ...
```

verwendet werden. Diese ist dann wahr, wenn die Referenz gültig ist (bzw. nicht gültig bei Angabe von NOT). Gültig ist die Referenz dann, wenn sie dereferenziert werden kann. Sie kennen bereits die Prüfung ... ref IS [NOT] INITIAL ... von der Prüfung von Objektreferenzen. Diese unterscheidet sich leicht von der hier gezeigten Prüfung. Für eine Objektreferenz ist IS BOUND wahr, wenn sie auf ein Objekt zeigt, und falsch, wenn Sie die Nullreferenz enthält. Bei einer Datenreferenz kann es jedoch dazu kommen, dass die Referenz zwar keine Nullreferenz mehr ist, sie jedoch trotzdem nicht dereferenziert werden kann. Dieses Phänomen tritt dann auf, wenn das referenzierte Datenobjekt aus dem Speicher entfernt wurde, etwa weil der Kontext, in dem das Datenobjekt definiert ist (Prozedur o.ä.) gelöscht wird.

Bislang haben Sie Datenobjekte erzeugt und anschließend mit `GET REFERENCE` eine Referenz darauf beschafft. Datenreferenzen bieten aber auch die Möglichkeit der direkten Erzeugung von Datenobjekten zur Laufzeit analog zur Objekterzeugung bei Objektreferenzen.

Auch die Syntax ist dabei ähnlich:

```
DATA ref TYPE REF TO type.  
CREATE DATA ref.
```

Hierdurch wird ein Datenobjekt des Typs `type` erzeugt, auf das die Referenz `ref` zeigt. Es ist auch die generische Typisierung möglich. Hierzu muss die Referenz generisch typisiert sein und der `CREATE DATA`-Anweisung ein Typ mitgegeben werden, der für die Erzeugung des Datenobjekts verwendet werden soll:

```
DATA ref TYPE REF TO data  
CREATE DATA ref TYPE type.
```

Dies ist auch dynamisch möglich:

```
DATA: ref TYPE REF TO data,  
      typename TYPE string VALUE 'SPFLI'.  
CREATE DATA ref TYPE (typename).
```

Datenreferenzen können einander zugewiesen werden. Auch hier werden Sie starke Parallelen zum Verhalten von Objektreferenzen erkennen. Durch Zuweisungen zwischen generisch und nicht-generisch typisierten Datenreferenzen kommt es auch hier zu entsprechenden Casts bei der Zuweisung.

```
DATA: ref_generic TYPE REF TO data,  
      ref_int TYPE REF TO i,  
      ref_date TYPE REF TO d,  
      int TYPE i VALUE 10,  
      dat TYPE d VALUE '20010101'.
```

```
GET REFERENCE OF int INTO ref_int.
```

```
ref_generic = ref_int.
```

```
ref_int ?= ref_generic.
```

Zuweisung möglich,
Da `ref_generic` auf
Datenobjekt des Typs `i` zeigt

```
GET REFERENCE OF dat INTO ref_date.
```

```
ref_generic = ref_date.
```

```
ref_int ?= ref_generic.
```

⚡ Laufzeitfehler durch
Typkonflikt

Abbildung 51: Zuweisung von Datenreferenzen

In der Abbildung wird zunächst die spezieller typisierte Datenreferenz `ref_int` der generisch typisierten Datenreferenz `ref_generic` zugewiesen. Diese Zuweisung funktioniert immer.

Die folgende Zuweisung funktioniert hingegen nur, weil `ref_generic` gerade auf ein entsprechend typisiertes Datenobjekt zeigt. Entsprechend führt die unterste Zuweisung zu einem Laufzeitfehler, da diese Voraussetzung nicht erfüllt ist: Der statische Typ der Zielreferenz muss dem dynamischen Typ der Quellreferenz entsprechen. Beide entsprechenden Zuweisungen sind mit dem `?=-`-Operator vorzunehmen, wie Sie es von Objektreferenzen kennen.

Beim zuweisen einer statisch typisierten Referenz zu einer generisch typisierten Referenz werden die Typeigenschaften übernommen. Dies gilt insbesondere auch für die Eigenschaften aus dem ABAP Dictionary. Beim Zuweisen einer statisch typisierten Datenreferenzvariablen zu einer generischen Datenreferenz gibt diese Ihre Typeigenschaften weiter. Dies soll am folgenden Übungsbeispiel gezeigt werden.

10.3 Praxis: Übung zur Typisierung von Datenreferenzen

Erstellen Sie ein neues Programm **ZZ_####_DREF** (ohne TOP-Include). Definieren Sie in diesem Programm ein Datenobjekt **cityfrom** welches Sie über die gleichnamige Komponente der **SPFLI**-Tabelle typisieren und eine Datenreferenz, die Sie **ref_cityto** nennen und über die Komponente **cityto** der Tabelle typisieren. Definieren Sie außerdem eine generische Datenreferenz **ref_generic**.

Sorgen Sie dafür, dass die Datenreferenz auf das Datenobjekt zeigt, und weisen Sie dem Datenobjekt den Namen einer Stadt zu. Geben Sie dann die Stadt über die Datenreferenz und den Dereferenzierungs-Operator `->` aus. Ihr Code könnte nun etwa wie folgt aussehen:

```
DATA:
  cityfrom      TYPE spfli-cityfrom,
  ref_cityto    TYPE REF TO spfli-cityto,
  ref_generic   TYPE REF TO data.

cityfrom = 'FRANKFURT'.

GET REFERENCE OF cityfrom INTO ref_cityto.
WRITE ref_cityto->.*.
```

Abbildung 52: Erster Quellcode: SAP-System-Screenshot

Speichern, prüfen, aktivieren und testen Sie das Programm. Klicken Sie in der Ausgabe auf den Namen der Stadt, so dass sich der Cursor darauf befindet, und drücken Sie **F1**.



Abbildung 53: F1-Hilfe: SAP-System-Screenshot

Sie sehen, dass die Hilfe zur Ankunftsstadt angezeigt wird. Verwendet wird also der Typ der Datenreferenz, nicht der Typ des referenzierten Datenobjekts.

Weisen Sie nun die typisierte Datenreferenz der generischen Datenreferenz zu. Geben Sie dann den Inhalt der generischen Datenreferenz aus. Da keine direkte Dereferenzierung möglich ist, definieren Sie hierzu zunächst ein (generisch typisiertes) Feldsymbol `<fs1>` und geben den Inhalt darüber aus. Ihr Quellcode könnte in diesem Abschnitt etwa wie folgt aussehen:

```
ref_generic = ref_cityto.
FIELD-SYMBOLS <fs1> TYPE ANY.
ASSIGN ref_generic->* TO <fs1>.
WRITE: <fs1>.
```

Abbildung 54: Zuweisung zur generischen Datenreferenz und Ausgabe: SAP-System-Screenshot

Speichern, prüfen, aktivieren und testen Sie das Programm erneut. Sie werden feststellen, dass auch auf der zweiten Ausgabe der Stadt die F1-Hilfe Informationen zur Ankunftsstadt anzeigt.



Abbildung 55: F1-Hilfe auf der zweiten Ausgabe: SAP-System-Screenshot

Öffnen Sie nun wieder den Quelltext. Holen Sie nun über den `GET REFERENCE OF` Befehl eine Referenz auf das Datenobjekt in die generische Referenzvariable. Definieren Sie ein weiteres Feldsymbol mit dem Namen `<fs2>` und geben Sie darüber den Inhalt der generischen Datenreferenz aus. Der Quelltext dieses Teils könnte wie folgt aussehen:

```
GET REFERENCE OF cityfrom INTO ref_generic.
FIELD-SYMBOLS <fs2> TYPE ANY.
ASSIGN ref_generic->* TO <fs2>.
WRITE: <fs2>.
```

Abbildung 56: Direktes holen der Referenz: SAP-System-Screenshot

Speichern, prüfen, aktivieren und testen Sie auch diesmal das Programm. Die F1-Hilfe auf der dritten Ausgabe der Stadt liefert ein anderes Ergebnis:



Abbildung 57: F1-Hilfe auf der dritten Ausgabe: SAP-System-Screenshot

Hier wird der Typ **cityfrom** zugrundegelegt, mit dem das Datenobjekt typisiert wurde.

10.4 Praxis: Übung zur Ausgabe von Tabelleninhalten

In dieser Übung werden Sie ein Programm schreiben, mit dem der Inhalt einer beliebigen, vom Benutzer eingegebenen Tabelle ausgegeben werden kann. Aufgrund der komplizierten Syntax finden Sie am Ende der Übung eine Musterlösung. Sollten Sie an einer Stelle nicht weiterkommen, können Sie dort nachschauen.

Nennen Sie ihr Programm **ZZ_####_TAB**. Für die Ausgabe der Tabelle wird wie gewohnt ein Arbeitsbereich benötigt. In diesem Fall kann dieser jedoch nicht einfach definiert werden, da der benötigte Typ zur Compilezeit nicht bekannt ist (dieser wird schließlich erst durch den Anwender festgelegt). Der Arbeitsbereich muss daher über eine Datenreferenz erzeugt werden. Definieren Sie diese Datenreferenz (Name: **ref_struct**) als generische Datenreferenz und einen Parameter zur Eingabe des Tabellennamens (Name: **pa_tab**).

Fügen Sie unterhalb der Definitionen **START-OF-SELECTION** ein. In diesem Block soll nun der Arbeitsbereich erzeugt werden. Verwenden Sie hierzu die **CREATE DATA**-Anweisung mit dynamischer Typisierung durch den vom Anwender eingegebenen Parameter. Bekanntlich können generische Datenreferenzen nicht direkt Dereferenziert und verwendet werden. Fügen Sie im Bereich der Definitionen daher die Definition eines Feldsymbols **<fs_struct>** hinzu, das generisch typisiert ist. Sorgen Sie dafür dass nach der Erzeugung des Datenobjekts (**CREATE DATA**) dieses von Ihrem Feldsymbol referenziert wird (verwenden Sie den **ASSIGN**-Befehl).

Sie können nun die Daten aus der Datenbank lesen. Fügen Sie daher nun eine **SELECT-ENDSELECT**-Schleife hinzu, in der Sie die Tabelle im **FROM**-Teil dynamisch als (**pa_tab**) und im **INTO**-Teil das Feldsymbol angeben. Ergänzen Sie zusätzlich **UP TO 50 ROWS** um die Ausgabe auf maximal 50 Zeilen zu beschränken.

Innerhalb eines Schleifendurchlaufs müssen nun die einzelnen Komponenten ausgegeben werden. Ihnen sind hier weder die Typen noch die Anzahl der Komponenten bekannt. Definieren Sie daher zunächst ein weiteres Feldsymbol **<fs_comp>**, um die jeweilige Komponente referenzieren zu können. Fügen Sie dann eine **DO...ENDDO**-Schleife in die **SELECT...ENDSELECT**-Schleife ein. In dieser soll die Struktur nun durchlaufen werden. Erinnern Sie sich an Abbildung 47 (Seite 45): Über einen Index kann auf die Komponenten einer Struktur mit **ASSIGN COMPONENT** zugegriffen werden. Als Index verwenden Sie hier einfach den Schleifenindex **sy-index**. Werten Sie anschließend den **sy-subrc** aus. Wenn

dieser ungleich 0 ist, wurde das Ende der Struktur erreicht. Erzeugen Sie in diesem Fall einen Zeilenwechsel mit `NEW-LINE` und verlassen Sie die Schleife mit `EXIT`. Geben Sie ansonsten die Komponente aus.

Prüfen Sie bei Bedarf nach `ENDSELECT` den `sy-subrc`, um dem Anwender mitteilen zu können, ob die Tabelle keine Daten enthielt.

Speichern, prüfen, aktivieren und testen Sie das Programm. Geben Sie als Tabellennamen **SPFLI** an und schauen Sie sich das Ergebnis an.



Programm ZZ_..._TAB							
703	AA	0017	US	NEW YORK	JFK US	SAN FRANCISCO	SFO 6:0
703	AA	0064	US	SAN FRANCISCO	SFO US	NEW YORK	JFK 5:2
703	AZ	0555	IT	ROME	FCO DE	FRANKFURT	FRA 2:0
703	AZ	0788	IT	ROME	FCO JP	TOKYO	TYO 12:5
703	AZ	0789	JP	TOKYO	TYO IT	ROME	FCO 15:4

Abbildung 58: Ausgaben des Programms: SAP-System-Screenshot

Wiederholen Sie den Test mit anderen Tabellen des Flugdatenmodells.

Im Folgenden sehen Sie ein Beispiel für einen entsprechenden Quellcode.

```

REPORT ZZ_#####_TAB.

DATA ref_struct TYPE REF TO data.

FIELD-SYMBOLS:
    <fs_struct> TYPE ANY,
    <fs_component> TYPE ANY.

PARAMETERS pa_tab TYPE c LENGTH 15.

START-OF-SELECTION.

CREATE DATA ref_struct TYPE (pa_tab).
ASSIGN ref_struct->* TO <fs_struct>.

SELECT * FROM (pa_tab) INTO <fs_struct> UP TO 50 ROWS.
DO.
    ASSIGN COMPONENT sy-index OF STRUCTURE <fs_struct> TO <fs_component>.
    IF sy-subrc <> 0.
        NEW-LINE.
        EXIT.
    ENDIF.
    WRITE <fs_component>.
ENDDO.
ENDSELECT.

IF sy-subrc <> 0.
    MESSAGE 'Leere Tabelle!' TYPE 'E'.
ENDIF.

```

Abbildung 59: Beispiel für eine Lösung der Aufgabe: SAP-System-Screenshot

10.5 Run Time Type Services (RTTS)

Unter den Run Time Type Services werden Möglichkeiten zur Abfrage von Typinformationen und Erzeugung von Typen zur Laufzeit verstanden. Die Abfragemöglichkeiten sind bereits seit Release 4.6C möglich und werden als RTTI (Run Time Type Identification) bezeichnet. Die dynamische Typerstellung zur Laufzeit (RTTC, Run Time Type Creation) kam erst mit NetWeaver 2004 zum SAP-System hinzu.

10.5.1 Run Time Type Identification (RTTI)

Das RTTI-Konzept ist objektorientiert umgesetzt und erstreckt sich auf sämtliche Typen. Es ersetzt damit die früher für diese Zwecke verwendeten Anweisungen `DESCRIBE FIELD` und `DESCRIBE TABLE`.

RTTI beschreibt Typen über sog. **Beschreibungsklassen**. Jeder Typ ist einer dieser Beschreibungsklassen zugeordnet, die dann entsprechende Attribute zur Beschreibung des Typs besitzt. Die Klassen zur Beschreibung von Referenzen, Strukturen, Tabellentypen, Klassen und Interfaces bieten Methoden an, um die jeweiligen Teiltypen (z. B. Komponenten einer Struktur) zu erreichen, so dass zu allen Teiltypen eines Typen navigiert werden kann. Die Beschreibungsklassen bilden eine Hierarchie, die in der folgenden Abbildung dargestellt wird. Die Hierarchie dieser Klassen entspricht der Hierarchie der Datentypen in ABAP.

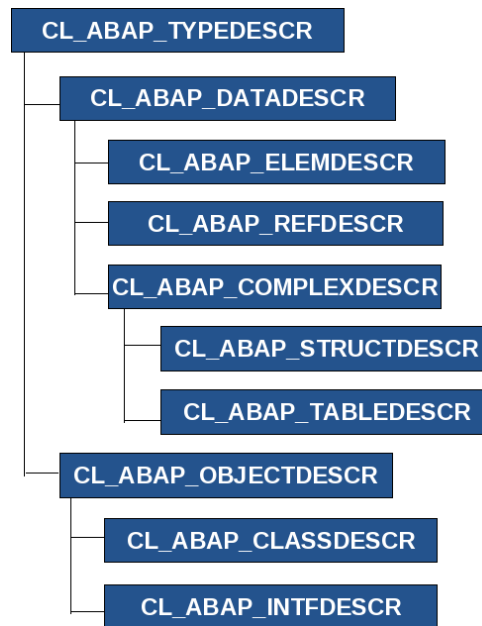


Abbildung 60: Hierarchie der RTTI-Beschreibungsklassen

Zu einem Typ kann mithilfe statischer Methoden der Klasse **CL_ABAP_TPEDESCR** eine Referenz auf ein zugehöriges **Beschreibungsobjekt** geholt werden. Alternativ kann ggf. über die Methoden zum Zugriff auf Teiltypen zum entsprechenden Typ navigiert werden, wenn dieser als Teiltyp auftritt und ein umgebender Typ bereits entsprechend referenziert wird. Zur Laufzeit existiert zu jedem Typ genau ein Beschreibungsobjekt.

Die statischen Methoden der Klasse **CL_ABAP_TPEDESCR** zum Holen des Beschreibungsobjekts sind die folgenden:

- **DESCRIBE_BY_DATA** holt das Beschreibungsobjekt indem ein Datenobjekt übergeben wird
- **DESCRIBE_BY_DATA_REF** holt das Beschreibungsobjekt indem eine Datenreferenz übergeben wird
- **DESCRIBE_BY_NAME** holt das Beschreibungsobjekt indem der Name des entsprechenden Typen übergeben wird
- **DESCRIBE_BY_OBJECT_REF** holt das Beschreibungsobjekt indem eine Objektreferenz übergeben wird

Des Weiteren besitzt die Klasse einige öffentliche Attribute, die an die entsprechenden Unterklassen vererbt werden und so in allen Beschreibungsklassen vorhanden sind. Dies sind insbesondere der absolute Typname **absolute_name**, der zugrunde liegende interne ABAP-Typ **type_name** und die interne Länge **length**.

Am folgenden Übungsbeispiel wird die Verwendung vorgestellt.

10.5.2 Praxis: Übung zur Ausgabe der Komponenten einer Struktur

Erstellen Sie für diese Übung ein weiteres Programm mit dem Namen **ZZ_####_SDESC** (ohne TOP-Include). Mit diesem Programm soll der Anwender die Komponenten einer Struktur anzeigen können.

Definieren Sie zunächst einen Parameter **pa_name** zur Eingabe des Namens der Struktur sowie eine Objektreferenz **r_descr**, mit der die Beschreibung referenziert werden soll. Verwenden Sie zur Typisierung die Klasse **cl_abap_structdescr**. Da in der Übung die Komponenten der Struktur durchlaufen werden, wird noch ein entsprechendes Datenobjekt benötigt. Die Strukturkomponenten werden in der Klasse **cl_abap_structdescr** einer internen Tabelle mit dem Zeilentyp **abap_compdescr** festgehalten. Verwenden Sie diesen Zeilentyp zur Typisierung und nennen Sie das Datenobjekt **gs_comp**.

Holen Sie nun die Referenz auf das Beschreibungsobjekt. Dies gelingt, indem Sie den Rückgabewert eines Aufrufs

```
cl_abap_typedescr=>describe_by_name( pa_name )
```

der Referenzvariablen **r_descr** zuweisen. Sie benötigen hierbei den Operator **?=>**, da der Rückgabewert vom Typ einer anderen Klasse ist.

Für die Ausgabe müssen Sie nun lediglich noch mit einer LOOP-Schleife die einzelnen Komponenten von **r_descr->components** in **gs_comp** zu lesen, und jeweils **gs_comp-name** auszugeben.



Abbildung 61: Ausgabe des Programms: SAP-System-Screenshot

Speichern, prüfen und aktivieren Sie das Programm. Testen Sie es anschließend und geben Sie als Strukturname SPFLI oder eine andere Struktur an. Die Abbildung zeigt die Ausgabe des Programms für SPFLI.

10.5.3 Praxis: Übung zur Ausgabe von Informationen über Klassen

In dieser Übung werden Sie mit RTTI Informationen über Klassen ermitteln. Erstellen Sie hierzu ein neues Programm mit dem Namen **ZZ_####_CDESC** (ohne TOP-Include). Binden Sie in dieses Programm mit der **INCLUDE**-Anweisung Ihr Include **ZZ_####_COMPANY_2_CLASSES** ein. Definieren Sie je eine Referenzvariable **r_employee** vom Typ der Mitarbeiterklasse und **r_office_employee** vom Typ der Innendienstmitarbeiterklasse.

Fügen Sie einen `START-OF-SELECTION`-Block ein und erzeugen Sie über die soeben definierte Referenzvariable `r_office_employee` ein Objekt der Innendienstmitarbeiterklasse und weisen Sie dieses anschließend der Referenzvariablen `r_employee` zu. Im Folgenden wird diese Referenz betrachtet. Mit RTTI soll nun ermittelt werden, welcher Klasse das referenzierte Objekt angehört. Definieren Sie daher eine Referenzvariable `r_desc` vom Typ `cl_abap_classdescr`. Diese soll eine entsprechende Typbeschreibung referenzieren. Sie erhalten diese als Rückgabewert eines Methodenaufrufs der Form

```
cl_abap_typedescr=>describe_by_object_ref( r_employee ).
```

Diese Methode nimmt eine Referenzvariable entgegen und liefert das Beschreibungsobjekt. Verwenden Sie auch hier zur Zuweisung zu `r_desc` den `==`-Operator.

Aus dem Beschreibungsobjekt soll nun der Name der Klasse ermittelt und ausgegeben werden. Definieren Sie daher eine String-Variable `name` und weisen Sie ihr den Klassennamen zu, den Sie als Rückgabewert der Methode `get_relative_name` des Beschreibungsobjekts erhalten. Geben Sie den Namen aus. Speichern, prüfen, aktivieren und testen Sie das Programm.

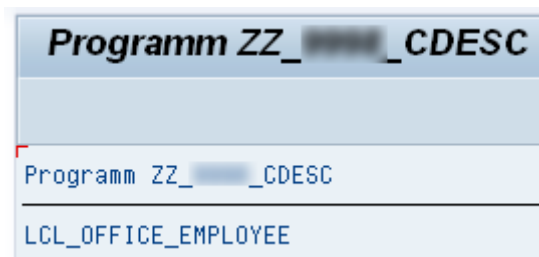


Abbildung 62: Ausgabe des Klassennamens: SAP-System-Screenshot

Sie sehen, dass die Klasse `lcl_office_employee` ausgegeben wird, also die Klasse der das Objekt angehört, und nicht etwa der statische Typ der Referenzvariablen `r_employee`.

Als nächstes soll mittels RTTI der Name der Oberklasse ausgegeben werden. Hierfür bietet das Beschreibungsobjekt der Klasse eine Methode `get_super_class_type`, die das Beschreibungsobjekt der jeweiligen Oberklasse zurückliefert. Holen Sie sich dieses Objekt, indem Sie eine weitere Referenzvariable `r_desc_super` des Typs `cl_abap_classdescr` definieren und ihr den Rückgabewert der Methode zuweisen. Lesen Sie auch aus diesem Objekt den Klassennamen aus und geben Sie ihn aus. Speichern, prüfen, aktivieren und testen Sie das Programm.

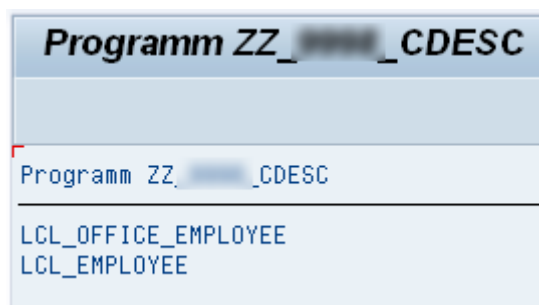


Abbildung 63: Ausgabe des Oberklassennamens: SAP-System-Screenshot

Wiederholen Sie die letzten Schritte, um auch den Namen der nächsten Oberklasse zu ermitteln und auszugeben.

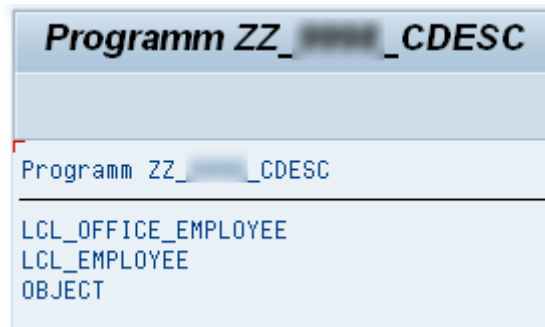


Abbildung 64: Ausgabe eines weiteren Oberklassennamens: SAP-System-Screenshot

Die Klasse **object** ist die allgemeinste Oberklasse des Systems. Ein weiteres Navigieren zu einer darüber liegenden Klasse ist daher nicht möglich, ein entsprechender Versuch würde zu einem Laufzeitfehler führen.

Schauen Sie sich bei Bedarf die Klasse **cl_abap_classdescr** im Class Builder an, um zu erfahren welche weiteren Methoden angeboten werden.

Die hier gezeigte Ermittlung des Klassennamens kann dazu genutzt werden, ohne Risiko eines Fehlers ein Casting auf eine speziellere Klasse durchführen zu können.

10.5.4 Run Time Type Creation (RTTC)

Seit NetWeaver 2004 ist nicht nur die Analyse von Typen möglich, sondern auch deren explizite Erzeugung mittels Run Time Type Creation (RTTC).

Zum Einsatz kommen hier dieselben Klassen, die Sie bereits von RTTI kennen. Diese besitzen statische Methoden, mit denen Typen der jeweiligen Klasse erzeugt werden können.

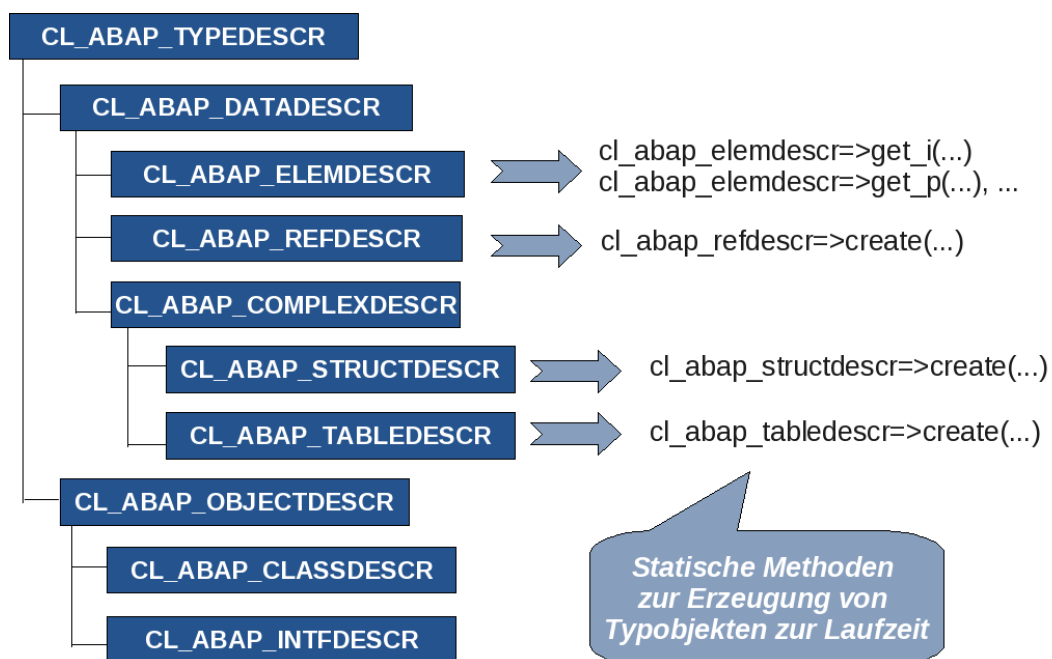


Abbildung 65: Methoden zur Erzeugung von Typobjekten

Mittels RTTC können sowohl elementare als auch komplexe Typen (z. B. interne Tabellen) erzeugt werden. Die Typobjekte sind transient, programmlokal und namenlos. Sie können weder gelöscht noch verändert werden.

10.6 Praxis: Übung zur Typerzeugung zur Laufzeit

In dieser Übung werden Sie einen Tabellentyp für interne Tabellen mit dem Zeilentyp SPFLI zur Laufzeit erstellen. Legen Sie hierfür ein neues Programm **ZZ_####_RTTC** (ohne TOP-Include) an. Zur Erzeugung des Typen benötigen Sie eine entsprechende Referenzvariable **r_table_type** des Typs **cl_abap_tabledescr**.

Öffnen Sie als nächstes im Navigationsbaum auf der linken Seite des Object Navigators diese Klasse, indem Sie aus dem Drop-Down-Feld oberhalb des Navigationsbaums **Klasse / Interface** wählen und den Klassennamen in das Eingabefeld darunter eingeben. Klappen Sie den Ast **Methoden** aus. Sie finden dort die **create**-Methode. Klicken und ziehen Sie diese in Ihren Quellcode auf der rechten Seite, so dass ein entsprechender Methodenaufruf generiert wird.

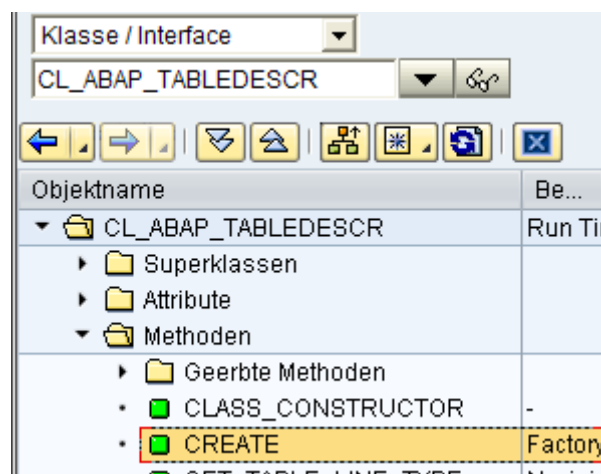


Abbildung 66: CREATE-Methode im Navigationsbaum: SAP-System-Screenshot

Der Methodenaufruf erwartet zunächst die Angabe eines Zeilentyps. Definieren Sie hierfür eine weitere Referenzvariable **r_line_type** vom Typ **cl_abap_structdescr**. Holen Sie durch die folgende Anweisung das Beschreibungsobjekt für die SPFLI-Struktur:

```
r_line_type ?= cl_abap_typedescr=>describe_by_name('SPFLI').
```

Übergeben Sie **r_line_type** dem Formalparameter **p_line_type**. Der Tabellentyp soll eine sortierte Tabelle beschreiben. Dies legen Sie fest, indem Sie dem Formalparameter **p_table_kind** den Wert

```
cl_abap_tabledescr=>tablekind_sorted
```

übergeben.

Um die Schlüsselfelder des Tabellentypen festzulegen, wird der Parameter **p_key** verwendet. Dieser erwartet eine interne Tabelle vom Typ **abap_keydescr_tab**. Definieren Sie eine solche Tabelle mit dem Namen **key**. Die Schlüsselfelder fügen Sie wie folgt der Tabelle hinzu:

```
APPEND 'CARRID' TO key.
APPEND 'CONNID' TO key.
```

Übergeben Sie **key** an den Formalparameter **p_key** und **abap_true** an den Formalparameter **p_unique**. Sorgen Sie anschließend noch dafür, dass das Ergebnis des Methodenaufrufs an **r_table_type** übergeben wird.

Der Typ ist nun erzeugt und kann für ein Datenobjekt verwendet werden. Definieren Sie hierfür eine generische Datenreferenz mit dem Namen **r_itab**. Um nun zu dieser Datenreferenz ein Datenobjekt zu erzeugen, das über den Typ verfügt, der durch das soeben erzeugte Beschreibungsobjekt repräsentiert wird, verwenden Sie folgende Syntax:

```
CREATE DATA r_itab TYPE HANDLE r_table_type.
```

Durch die Angabe des Type Handle wird nun eine interne Tabelle gemäß der Typbeschreibung erzeugt. In diese Tabelle sollen nun die Inhalte der Datenbanktabelle SPFLI gelesen werden. Definieren Sie hierzu ein Feldsymbol **<fs>**, welches Sie mit **ANY TABLE** typisieren. Stellen Sie mit **ASSIGN** die Referenz her und lesen Sie die Daten durch das Feldsymbol in die interne Tabelle:

```
FIELD-SYMBOLS <fs> TYPE ANY TABLE.
ASSIGN r_itab->* TO <fs>.
SELECT * FROM spfli INTO TABLE <fs>.
```

Ergänzen Sie am Ende des Programms noch einen statischen Breakpoint (**BREAK-POINT.**) Dieser soll hier dazu dienen, das erzeugte Datenobjekt im Debugger betrachten zu können bevor der Programmkontext verlassen wird. Eine Ausgabe der gelesenen Daten ist in dieser Aufgabe nicht vorgesehen.

Speichern, prüfen und aktivieren Sie das Programm. Testen Sie das Programm anschließend. Klicken Sie im Debugger doppelt auf **r_itab->*** in Ihrem Quelltext. Es erscheint daraufhin ein entsprechender Eintrag auf der rechten Seite:

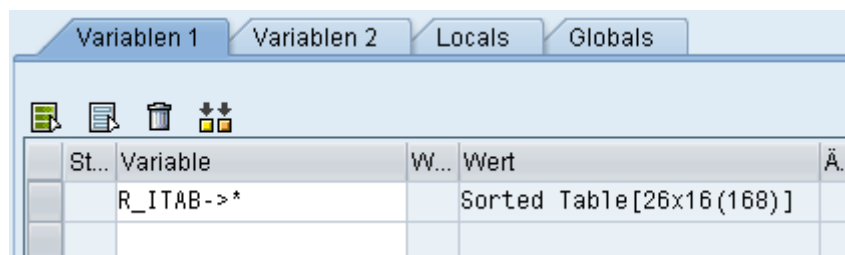


Abbildung 67: Variablenanzeige im Debugger: SAP-System-Screenshot

Klicken Sie dort ebenfalls doppelt auf **R_ITAB->***. Sie sehen daraufhin den Aufbau und Inhalt der internen Tabelle.

Tabellen		Tabelleninhalt					
Tabelle		R_ITAB->*					
Tabellentyp		Sorted Table[26x16(168)]					
	Zeile	MANDT[C(3)]	CARRID[C(3)]	CONNID[N(4)]	COUNTRYFR[C(3)]	CITYFROM[C(20)]	AIRPFROM[C(3)]
	1	703	AA	0017	US	NEW YORK	JFK
	2	703	AA	0064	US	SAN FRANCISCO	SFO
	3	703	AZ	0555	IT	ROME	FCO
	4	703	AZ	0788	IT	ROME	FCO
	5	703	AZ	0789	JP	TOKYO	TYO
	6	703	AZ	0790	IT	ROME	FCO
	7	703	DL	0106	US	NEW YORK	JFK
	8	703	DL	1699	US	NEW YORK	JFK
	9	703	DL	1984	US	SAN FRANCISCO	SFO

Abbildung 68: Anzeige der Tabelle im Debugger: SAP-System-Screenshot

Sie haben nun RTTC genutzt, um zur Laufzeit einen Tabellentypen zu erzeugen und diesen Typen zur Erzeugung eines Datenobjekts verwendet. Der Effekt mag bei diesem Beispiel noch nicht besonders beeindruckend sein, da in diesem Programm auch einfach statische Typen hätten verwendet werden können. Die dynamische Typerzeugung eröffnet aber die Möglichkeit beliebige Typen zu definieren, ohne dass diese zur Kompilierzeit bekannt sein müssen. So hätte hier statt dem Rückgriff auf das Beschreibungsobjekt von SPFLI eine eigene Struktur über die Mechanismen von RTTC erzeugt werden können, die nur bestimmte Felder enthält. Der hohen Flexibilität solcher dynamischer Typen steht jedoch ein gewisser syntaktischer Aufwand gegenüber. Beides ist gegeneinander abzuwägen.

10.7 Kontrollfragen

1. Ändert ein schreibender Zugriff auf ein Feldsymbol die Speicheradresse, auf die das Feldsymbol zeigt, oder den Inhalt, also die eigentlichen Daten?
2. Gibt eine statisch typisierte Datenreferenzvariable bei Zuweisung zu einer generischen Datenreferenzvariablen ihre Typeigenschaften weiter?
3. Durch welche syntaktische Besonderheit sind Feldsymbole zu erkennen?
4. Können Datenobjekte auch erst zur Laufzeit typisiert werden?
5. Kann eine mit `REF TO data` typisierte Datenreferenz einer mit `REF TO i` typisierten Datenreferenz zugewiesen werden?
6. Was wird durch den `CASTING`-Zusatz des `ASSIGN`-Befehls bewirkt?
7. Wie kann auf ein Datenobjekt zugegriffen werden, wenn nur eine mit `REF TO data` typisierte Datenreferenz zu diesem Datenobjekt zur Verfügung steht?

10.8 Antworten

1. Es wird der Inhalt des Feldsymbols überschrieben.
2. Ja
3. Sie stehen immer in spitzen Klammern.
4. Ja
5. Wenn der dynamische Typ stimmt ist die Zuweisung möglich, es ist aber Grundsätzlich der Operator `?=` zu verwenden.
6. Der Speicherbereich so behandelt, als hätte er den durch `CASTING` angegebenen Typ.
7. Der Zugriff erfolgt über ein Feldsymbol, da eine direkte Dereferenzierung nicht möglich ist.

11 System- und Unternehmensübergreifende Geschäftsprozesse

Nicht immer sind die IT-Systeme in einem Unternehmen vollständig integriert. Dies kann verschiedene Gründe haben, etwa verteilte Standorte, organisatorische Überlegungen oder historisch gewachsene IT-Architekturen. So könnte es etwa vorkommen, dass eine Abteilung eines Unternehmens ein separates System verwendet, aber dennoch im Rahmen der Geschäftsprozesse des Unternehmens auf einen Austausch mit den Systemen anderer Abteilungen angewiesen ist.

Auch zwischen Unternehmen ist häufig ein Austausch von Daten zwischen den jeweils verwendeten Systemen vonnöten. Unternehmen, die eng kooperieren, verzahnen ihre Geschäftsprozesse, um gemeinsam ihre Ziele besser erreichen zu können. Ein denkbare Szenario wäre etwa ein gemeinsamer Einkauf bei einem Zulieferer. System- und Unternehmensübergreifende Prozesse treten immer öfter auf und müssen daher durch die IT-Systeme adäquat unterstützt werden.

11.1 Application Link Enabling

SAP bietet für den Aufbau und Betrieb verteilter Anwendung eine spezielle Middleware-Technologie an. Diese wird als **Application Link Enabling (ALE)** bezeichnet. Mit ALE wird eine Systemlandschaft ermöglicht die zwar verteilt, aber dennoch integriert ist. Genauer gesagt besteht eine lose Kopplung der Systeme, die über Nachrichten miteinander kommunizieren, wobei *betriebswirtschaftliche* Informationen übertragen werden. Die Kommunikation kann dabei synchron oder asynchron erfolgen. Die beteiligten Systeme verwalten ihre Daten jedoch unabhängig voneinander in ihrer jeweiligen Datenbank. Dabei ist zu gewährleisten, dass der Datenbestand der beteiligten Systeme sich stets in einem konsistenten Zustand befindet.

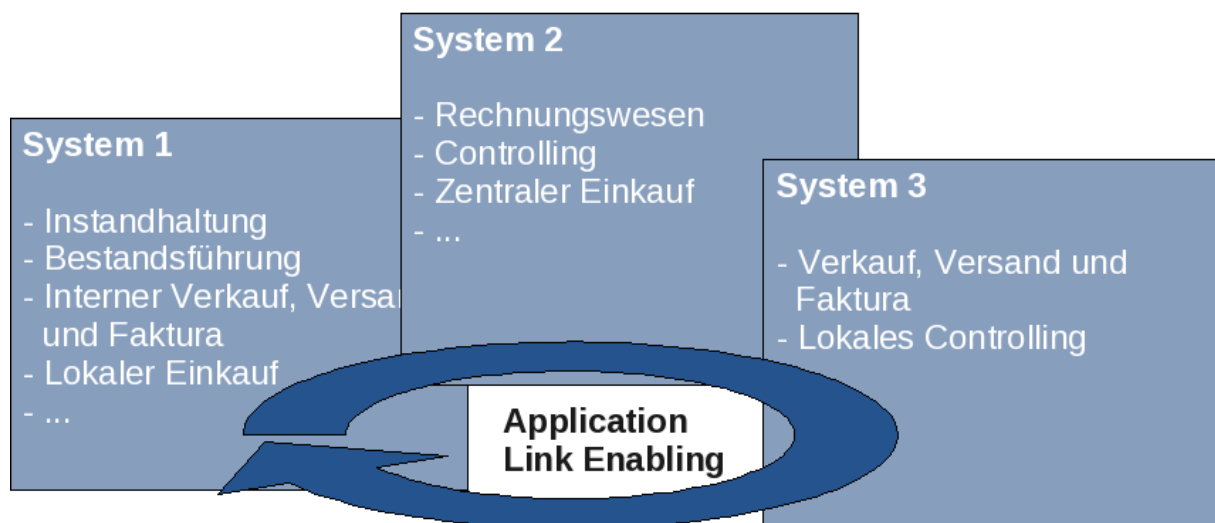


Abbildung 69: Anwendungsszenario für ALE

Die Aufgabe von ALE ist es, den Datenaustausch bezüglich der Kommunikationspartner, auszutauschenden Daten, Übertragungswege und -zeitpunkte zu koordinieren. Hierfür gilt es zu identifizieren, welcher Geschäftsprozess und welche Objekte für den Datenaustausch relevant sind, welche Informationen übertragen werden sollen, in welchem Format die Daten übertragen werden sollen, mit welcher Technik diese Übertragung realisiert werden soll,

welche Übertragungsart (synchron / asynchron) verwendet werden soll und welches Ziel mit der Datenübertragung verfolgt wird.

Als Beispiel kann der Fall eines Datenabgleichs von Bestellungen zwischen einem SAP CRM-System und einem SAP ERP-System herangezogen werden:

- Geschäftsprozess: Internet Sales mit SAP CRM
- Zu übertragende Informationen: Daten von Bestellungen aus dem CRM-System zur Weitergabe an das ERP-System
- Datenformat: IDoc
- Übertragungsart: Asynchrone Übertragung alle 120 Sekunden
- Ziel der Übertragung: Angebot von Waren oder Dienstleistungen im Internet

Die zu übertragenden Daten werden häufig in Form von zugeordneten Business Objects identifiziert, die wiederum über entsprechende BAPIs (Business Application Programming Interfaces) verfügen. Ein BAPI stellt eine Methode auf ein betriebswirtschaftliches Objekt dar, etwa das Ändern von Daten einer Bestellung. In der Regel können alle Daten des Business Objects über BAPIs bearbeitet werden.

IDoc ist ein Akronym für „Intermediate Document“. Es handelt sich hierbei um ein Datenaustauschformat von SAP, wobei es für verschiedene auszutauschende Daten entsprechende spezifische Formate gibt. Dieses Prinzip ähnelt dem von XML (Extensible Markup Language). Auch die Übertragung mit einem XML-Format ist möglich.

Auch die Wahl der Übertragungstechnik lässt verschiedene Optionen, wie etwa RFC (Remote Function Call), HTTP (Hypertext Transfer Protocol) oder HTTPS (Hypertext Transfer Protocol Secure).

Bei der synchronen Übertragungsart werden Daten sofort übertragen, wenn sie anfallen. Bei der asynchronen Übertragung erfolgt der Datenaustausch zu regelmäßigen Zeitpunkten.

Um per ALE verknüpft werden zu können, müssen die beteiligten Systeme das entsprechende Datenformat fachlich interpretieren können und die technischen Voraussetzungen zur Übertragung bieten, d. h. Übertragungstechniken und -Format unterstützen. Diese Hürde ist relativ gering, so können etwa SAP-Systeme unterschiedlicher Releasestände miteinander per ALE gekoppelt werden.

11.2 Business Objects und BAPIs

Business Objects befinden sich im SAP-System im **Business Object Repository** (BOR). In Business Objects werden Objekte der Geschäftswelt repräsentiert, die sich in einer Tabelle bzw. Tabellenhierarchie befinden. Mittels BAPIs kann auf die Business Objects zugegriffen werden. Im Kapitel zu Web Dynpro haben Sie ein BAPI verwendet, um Daten auszulesen. In der Regel werden alle Grundfunktionalitäten eines Business Objects durch BAPIs verfügbar gemacht, also neben dem Auflisten von Objekten deren Erzeugung sowie lesender und schreibender Attributzugriff. Es gibt hierfür BAPIs mit standardisierten Namen:

- GetList zur Auflistung von Objekten (ggf. mit Angabe von Filterkriterien)
- GetDetail zum Lesen der Attribute eines bestimmten Objekts (Angabe des vollständigen Schlüssels erforderlich)
- Create, Change, Delete und Cancel zum Erzeugen, Ändern und Löschen von Objekten
- AddItem bzw. RemoveItem zum Hinzufügen oder Entfernen von Teilobjekten, etwa eine Bestellposition auf einer Bestellung

Die Funktionalität wird durch remotefähige Funktionsbausteine gekapselt. Hierdurch ist ein systemübergreifender Zugriff auf die BAPIs gewährleistet. Über diese Technik ist z. B. auch ein Zugriff aus Nicht-SAP-Systemen, wie etwa eine in Visual Basic realisierte Anwendung des Kunden möglich. Die Suche nach BAPIs ist über den **BAPI-Explorer** (Transaktion **BAPI**) möglich.

Für BAPI-Funktionsbausteine gelten einige Bedingungen:

- Die Namenskonvention `BAPI_bobject_mname`, wobei `bobject` für den Namen des Business Objects und `mname` für dessen Methode steht, muss eingehalten werden
- Sie müssen remote-fähig sein (RFC-Schnittstelle).
- Sie dürfen keine Dialoge oder Nachrichten enthalten.
- Zur Typisierung der Schnittstellenparameter dürfen nur Komponenten spezieller Strukturen aus dem Dictionary verwendet werden. Das Namenspräfix hierfür lautet `BAPI_`.
- Bis Release 4.6 waren keine Changing-Parameter erlaubt.
- Sie dürfen keine Ausnahmen auslösen. Fehlersituationen werden stattdessen über einen speziellen Export-Parameter `RETURN` an den Aufrufer übergeben.
- Die Schnittstelle ist für externe Aufrufer konzipiert. Infolgedessen werden Beträge in einem externen Format mit 4 oder 9 Nachkommastellen erwartet, die beim Aufruf aus dem lokalen System ggf. konvertiert werden müssen. Hierfür stehen in der Funktionsgruppe `BACV` entsprechende Funktionsbausteine zur Verfügung.

Die RFC-Schnittstelle basiert auf CPI-C (Common Program Interface Communication) und TCP/IP (Transmission Control Protocol/Internet Protocol). Aus dem ABAP-Grundlagen-Kurs kennen Sie bereits die Syntax zum Aufruf entfernter Funktionsbausteine. Ein Vorteil von RFC ist die Unabhängigkeit von einer bestimmten Programmiersprache. Es wird lediglich eine Schnittstelle definiert, die von Anwendungen unterschiedlicher Sprachen verwendet werden kann. Insbesondere können auch Nicht-SAP-Systeme RFCs des SAP-Systems nutzen. Auch umgekehrt ist ein Zugriff möglich, so dass aus einem SAP-System z. B. eine Dynamic Link Library (DLL) als RFC-Schnittstelle genutzt wird. RFC kann also bidirektional genutzt werden. Auch R/2-Systeme unterstützen bei entsprechendem Release den Zugriff per RFC. Die technische Verbindung, über die die RFC-Kommunikation abläuft, wird **RFC-Verbindung** oder **RFC-Destination** genannt. Ohne eine solche Verbindung wäre die RFC-Nutzung nicht möglich. Die Verbindungen werden in der Transaktion **SM59** gepflegt.

Tipp: Im Laufe des Kurses haben Sie eine Vielzahl von Transaktionen kennen gelernt. Es mag vorkommen, dass Sie eine Transaktion einmal vergessen oder keinen Favoriten angelegt haben. Es ist dann aufgrund des Umfangs schwer, im SAP-Menü die entsprechende Transaktion wiederzufinden. Hilfreich sind hier die Transaktionen **SEARCH_SAP_MENU** und **SEARCH_USER_MENU**. Erstere durchsucht das SAP-Menü, während letztere das Benutzermenü durchsucht. Dabei handelt es sich um ein Menü, das abhängig von der im Benutzerstammsatz eingestellten Rollen des Benutzers generiert wird. In der Tabelle **USERS_SSM** können Administratoren festlegen, ob ein Benutzer zwischen diesen Menüs umschalten darf. Die Sichtbarkeit des Menüs beeinflusst aber nicht die Ausführbarkeit der enthaltenen Transaktionen durch Eingabe des Transaktionscodes in das Kommandofeld.

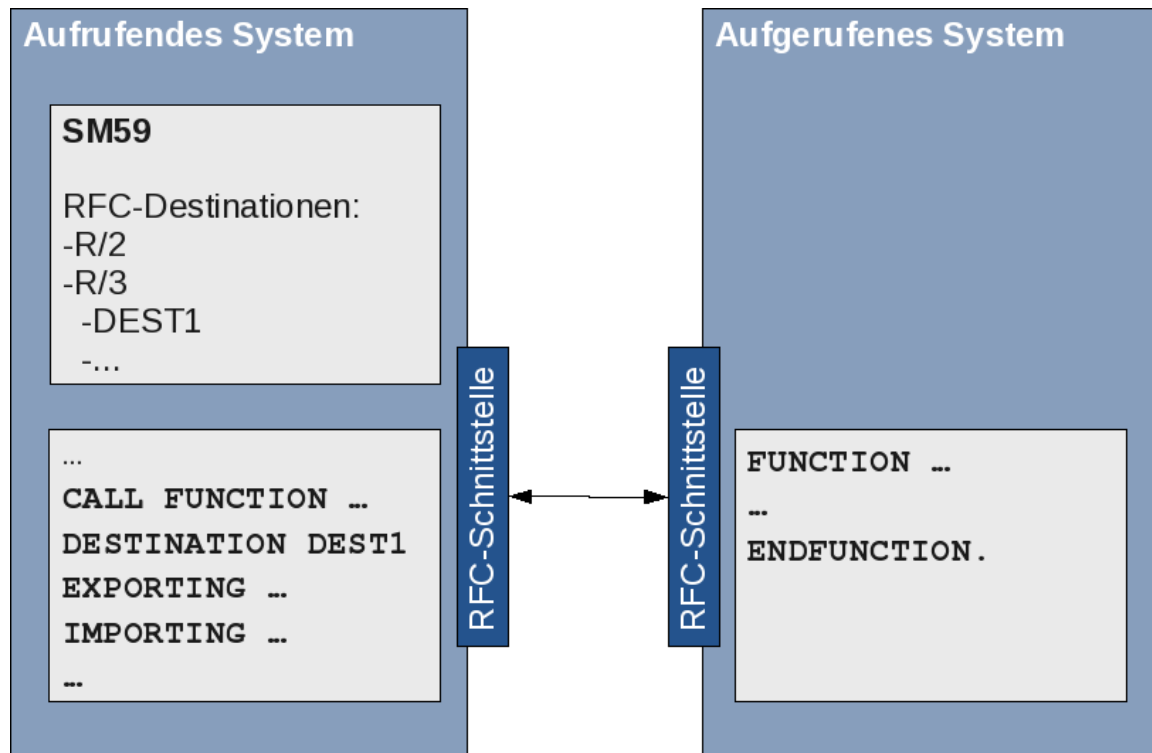


Abbildung 70: RFC-Aufruf eines Funktionsbausteins

Es ist zu betonen, dass eine RFC-Destination, also das Ziel eines RFC-Aufrufs, nicht ein ganzes System ist. Stattdessen besteht immer ein Bezug zu einem bestimmten Mandanten des Zielsystems (auf dem aufrufenden System steht sie aber allen Mandanten zur Verfügung). Es wird daher im ALE-Umfeld von **logischen Systemen** gesprochen. Es können mehrere RFC-Destinationen zu einem System und einem Mandanten bestehen, da zur Destination auch ein Benutzer zur Anmeldung im Zielsystem angegeben werden kann. So können Destinationen mit unterschiedlichen Benutzern definiert werden. Fehlt diese Angabe in der Destination, so ist sie beim Aufruf des Funktionsbausteins über entsprechende Parameter vorzunehmen. Eine alternative sind **Trusted RFCs**, bei denen der Benutzer verwendet wird, der im aufrufenden System gerade angemeldet ist. Dies setzt voraus, dass dieser Benutzer auch im Zielsystem vorhanden ist. Soll auch das gerufene System auf das aufrufende System zugreifen können, also eine bidirektionale Verbindung hergestellt werden, muss in diesem System ebenfalls eine entsprechende Destination angelegt werden.

Bei der Kommunikation zwischen Instanzen eines Systems oder zwischen Systemen per RFC oder CPIC wird stets das Gateway verwendet. Tritt etwa in einem Dialog-Workprozess ein entsprechender Aufruf auf, so wendet sich der Dialog-Workprozess an das Gateway. Dieses leitet die Anfrage an das Gateway des entfernten Systems, welches die Anfrage wiederum an den Dispatcher übergibt. Dieser leitet die Anfrage an einen seiner Workprozesse weiter, der dann direkt mit dem Gateway (des entfernten Systems) kommuniziert.

11.3 Praxis: Übung zu BAPIs

Öffnen Sie den BAPI-Explorer (Transaktion **BAPI**). Wählen Sie den Menüpfad **Bearbeiten->Suchen** und geben Sie als Suchbegriff **FLIGHT** ein:

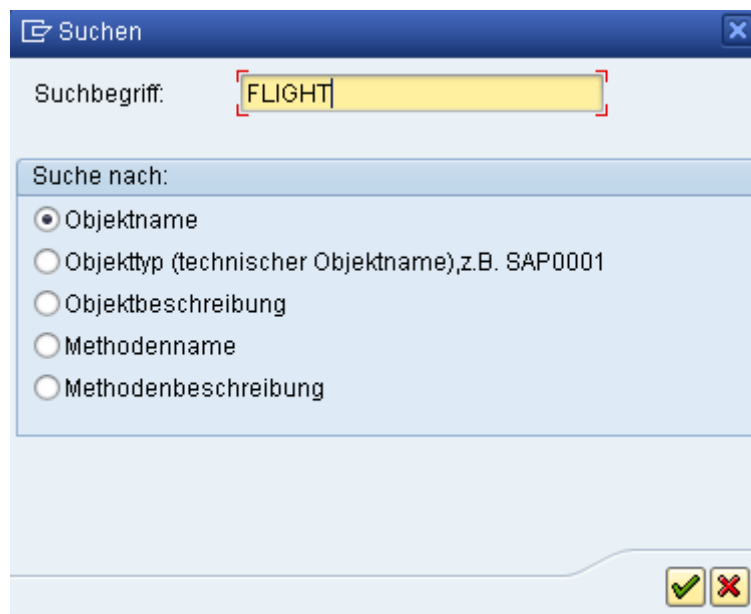


Abbildung 71: Suche im BAPI-Explorer: SAP-System-Screenshot

Sie gelangen daraufhin zum entsprechenden Business-Objekt. Wählen Sie auf der rechten Seite die Registerkarte **Dokumentation**, um sich eine Beschreibung des Objekts anzusehen. Wechseln Sie anschließend zur Registerkarte **Werkzeuge** und klicken Sie doppelt auf **BAPI-Liste erstellen**. Bestätigen Sie die Suche. Es wird daraufhin eine Liste der **BAPIs** angezeigt:

Objekttyp	Objektname	Methodenname	Name des Funktionsbausteins
SFLIGHT	Flight	CheckAvailability	BAPI_FLIGHT_CHECKAVAILABILITY
SFLIGHT	Flight	GetDetail	BAPI_FLIGHT_GETDETAIL
SFLIGHT	Flight	GetList	BAPI_FLIGHT_GETLIST
SFLIGHT	Flight	SaveReplica	BAPI_FLIGHT_SAVEREPLICA

Abbildung 72: Ergebnis der Suche: SAP-System-Screenshot

In den Ergebnissen finden Sie unter anderem die Methode GetList und den zugehörigen Funktionsbaustein, den Sie bereits verwendet haben. Weiterhin sehen Sie andere Methoden, die für dieses Objekt verfügbar sind.

Kehren Sie zur Standardansicht des BAPI-Explorers zurück. Klappen Sie im Navigationsbaum auf der linken Seite den Knoten Flight aus. Sie sehen dort ebenfalls die verfügbaren Methoden:

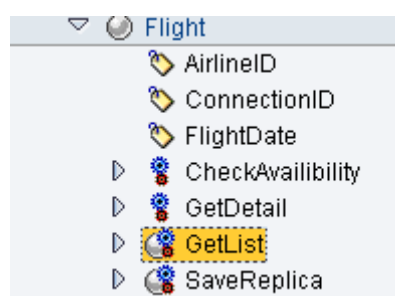


Abbildung 73: Navigationsbaum des BAPI-Explorers: SAP-System-Screenshot

Wählen Sie GetList aus. Auf der rechten Seite stehen nun nähere Informationen zum BAPI (auf der **Detail**-Registerkarte). Doppelklicken Sie dort auf den Namen des Funktionsbausteins. So navigieren Sie direkt zum Baustein im Function Builder. Sie können sich dort mit der Implementierung vertraut machen. Dort begegnet Ihnen eine etwas ungewohnte Syntax der Form `tabellename[]`. Diese Syntax stellt sicher, dass bei Tabellen mit Kopfzeile der Tabellenrumpf und nicht etwa nur die Kopfzeile angesprochen wird. Bei Tabellen ohne Kopfzeile besteht kein Unterschied zwischen Tabelle und Tabellenkörper. Weiterhin werden dort Befehle zum Löschen von Tabelleninhalten verwendet.

Es gibt drei solche Befehle: `REFRESH itab` löscht den Inhalt, ein Teil des zuvor reservierten Arbeitsspeichers bleibt jedoch für Einfügungen reserviert, `CLEAR itab` verhält sich bei normalen Tabellen genauso, initialisiert bei internen Tabellen mit Kopfzeile aber nur die Kopfzeile, `FREE itab` hingegen gibt nach dem Löschen den belegten Arbeitsspeicher komplett frei (dieser Befehl ist geeignet, wenn die Tabelle im weiteren Verlauf nicht mehr benötigt wird).

11.4 Enterprise SOA und Web Services

Die Softwaresysteme von SAP sind einer stetigen Weiterentwicklung unterworfen, um stets den aktuellen Anforderungen an die Systeme in möglichst hohem Maße gerecht zu werden. Eines der entscheidenden Merkmale früher Systeme war die Echtzeit-Verarbeitung, die SAP zu einem führenden Anbieter von Unternehmenssoftware gemacht hat. Sowohl SAP R/2 als auch SAP R/3 verfügten über dieses Merkmal. Beide Systeme setzten ABAP als spezielle Programmiersprache für betriebswirtschaftliche Anwendungen ein, boten einen wachsenden Funktionsumfang durch von SAP oder den Kunden entwickelte Funktionen, verwendeten eine zentrale Datenbank und konnten an die kundenspezifischen Prozesse angepasst werden. Als Kontrast zur hostbasierten Architektur von R/2 wurde mit R/3 ein Client-Server-Basiertes System eingeführt.

In der weiteren Entwicklung wurde die eigenständige, flexible Anwendungsplattform zu einem Kern des technischen Fortschritts im SAP-System. Allen zukünftigen SAP-Anwendungen steht mit der NetWeaver-Plattform eine einheitliche technische Grundlage zur Verfügung.

Mit SAP ERP 6.0 steht SAP an der Schwelle zu einer neuartigen Architektur: Der Enterprise Service Oriented Architecture (Enterprise SOA). Hauptansatzpunkt sind hier die Schnittstellen zwischen Anwendungen. In der klassischen Client-Server-Architektur werden etwa Nicht-SAP-Anwendungen mehr oder weniger aufwändig über Schnittstellen angebunden. Vielfach bestehen aber nicht einmal automatisierte, technische Schnittstellen, sondern Menschen („menschliche Integratoren“) übernehmen die Übertragung von Daten, indem sie die Systeme aufrufen und verwenden und so die Prozesse integrieren. Innerhalb des Systems findet häufig ein unkontrollierter Austausch von Daten zwischen den Anwendungen statt, da alle Anwendungen prinzipiell auf die Datenbank zugreifen können und entsprechende Tabellen bearbeiten können.

In der Enterprise SOA ist ein anderes Konzept vorgesehen: Hier würde die „fremde“ Anwendung, die auf Daten zugreifen möchte, die zu einem anderen Bereich des Systems gehören, diese über eine Application-to-Application-Schnittstelle (A2A-Schnittstelle) anfordern, anstatt selbst auf die Datenbank zuzugreifen.

Prozessschritte liegen in der Enterprise SOA als **Enterprise Services** vor. Durch die Verwendung dieser einzelnen Prozessschritte können systemübergreifende Prozesse definiert

werden und in rollenspezifischen Oberflächen den Mitarbeitern als zentraler Einstiegspunkt angeboten werden. Eine entsprechende Prozesssteuerung verringert den Aufwand für die „menschlichen Integratoren“. Die einzelnen Schritte werden über spezielle Schnittstellen miteinander verknüpft. Dies sind neben den oben erwähnten Application-to-Application-Schnittstellen zur Verknüpfung innerhalb von Geschäftsanwendungen Business-to-Business-Schnittstellen (B2B-Schnittstellen) und User Interface-Schnittstellen (UI-Schnittstellen). Die Schnittstellen stellen den Kern der Enterprise SOA dar. Geschäftsanwendungen sollen mithilfe dieser Schnittstellen top-down modelliert werden. So sollen schließlich jegliche Funktionalitäten des Systems aus dem Modell begründbar sein.

Zur technischen Realisierung der Schnittstellen werden standardisierte Protokolle verwendet: Hier kommt das Konzept der **Web Services** zum Einsatz. Mit diesen werden die Detailfunktionen, die innerhalb der Anwendung bzw. des Enterprise Services ablaufen bzw. aufgerufen werden umgesetzt. Der Enterprise Service stellt die umfassende Geschäftslogik dar, ohne dabei auf die Detailfunktionen abzielen. Für die Beschreibung und Verwendung von Web Services werden Standards verwendet:

- Web Service Description Language (WSDL): Mit dieser vom W3C (World Wide Web Consortium) standardisierten Sprache wird die Funktionalität des Web Service beschrieben. Die Beschreibung (Funktionen, Parameter und Rückgabewerte) ist maschinenlesbar. Die Sprache basiert auf XML. Nähere Informationen finden Sie unter <http://www.w3.org/TR/wsdl.html>
- SOAP (ehemals Akronym für Simple Object Access Protocol) ist das Protokoll das zum Aufruf der Web Services über http benutzt wird. SOAP-Nachrichten bestehen aus einem Kopf mit zusätzlichen Informationen und einem Rumpf mit der eigentlichen Nachricht.
- Universal Description, Discovery and Integration (UDDI) ist ein Verzeichnisdienst, in dem ein Web Service-Anbieter seine Services veröffentlichen kann. Für die Informationen werden sog. White-, Yellow- und Green Pages verwendet. SAP bietet einen UDDI-Server unter <http://uddi.sap.com> an. Weitere Informationen über UDDI finden Sie unter www.uddi.org.

Bereits ab Release 6.20 können Web Services definiert werden, seit Release 6.40 steht jedoch eine erheblich erweiterte Funktionalität zur Verfügung. Vorhandene, RFC-Fähige Funktionsbausteine können hiermit in Web Services umgewandelt werden. Die folgende Abbildung skizziert ein einfaches Web Service-Szenario aus Veröffentlichung, Suche und Ausführung.

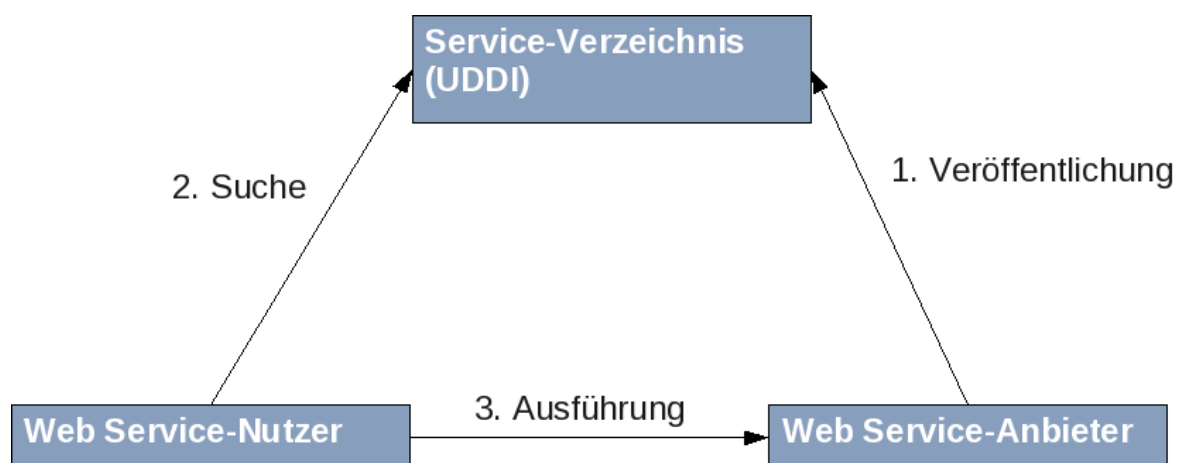


Abbildung 74: Nutzungsszenario von Web Services

Bei der Generierung des Web Services aus einem Funktionsbaustein werden eine URL und eine beschreibende WSDL-Datei generiert. Der Nutzer des Web Services legt ein Proxy-Objekt an, das auf die URL verweist. Eine passende ABAP-Klasse wird generiert und ein logischer Port zugeordnet. Das Proxy-Objekt wird vom Web Service-Nutzer dann in ein Programm eingebunden und der Service dort aufgerufen.

Enterprise SOA-Anwendungen werden in der Regel systemübergreifend in ABAP oder auch in Java und ohne eine „eigene“ Datenbank erstellt. Neue Funktionen werden außerhalb von bestehenden Systemen erstellt. Die Vorteile dieses Ansatzes bestehen darin, dass neue Anwendungen effizient erstellt werden können, ohne dass Anpassungen der zugrundeliegenden Systeme erforderlich werden, in einer hohen Flexibilität, die sich etwa in der Möglichkeit widerspiegelt im laufenden Betrieb Geschäftsprozesse konfigurieren zu können, und der vereinfachten Entwicklung von Anwendungen, für die Funktionen mehrerer Systeme benötigt werden.

Aufgrund dieser Vorteile strebt SAP einen starken Ausbau des Angebots an Enterprise Services an. So wird Kunden zudem die Möglichkeit geboten, Enterprise- und Web Services anderer Anbieter über das Internet in ihre Anwendungen zu integrieren.

Die in diesem Kapitel vorgestellten Protokolle sind nicht die einzigen Möglichkeiten aus einem SAP-System zu kommunizieren. Die Bandbreite der unterstützten Protokolle und Schnittstellen ist hoch, es werden etwa auch EDI (Electronic Data Interchange), LU 6.2 (Logical Unit Type 6.2), OLE (Object Linking and Embedding) und SMTP (Simple Mail Transfer Protocol) verwendet.

12 Der Code Inspector

Mit dem SAP Web Application Server 6.20 ist ein neues Testwerkzeug zum SAP-System hinzugekommen. Der **Code Inspector** ermöglicht verschiedene statische Prüfungen Ihres Programmcodes. Dabei werden stehen drei Kriterien im Mittelpunkt:

- **Performance:** Werden z. B. Indizes bei Datenbankzugriffen benutzt?
- **Sicherheit:** Wird z. B. auf andere Mandanten zugegriffen?
- **Semantik:** Werden typische Semantikfehler gemacht? Darunter fällt etwa eine Fehlende sy-subrc-Prüfung nach AUTHORITY-CHECK, eine fehlende Mandantenangabe bei einer SQL-Anfrage mit CLIENT SPECIFIED-Zusatz oder das senden mehrerer Nachrichten des Typs „E“ direkt hintereinander.

Sie erreichen den Code Inspector indem Sie mit der rechten Maustaste auf das zu prüfende Programm im Navigationsbaum des Object Navigator klicken und aus dem Kontextmenü **Prüfen->Code Inspector** wählen, oder aus dem Menü unter **Programm** demselben Pfad folgen.

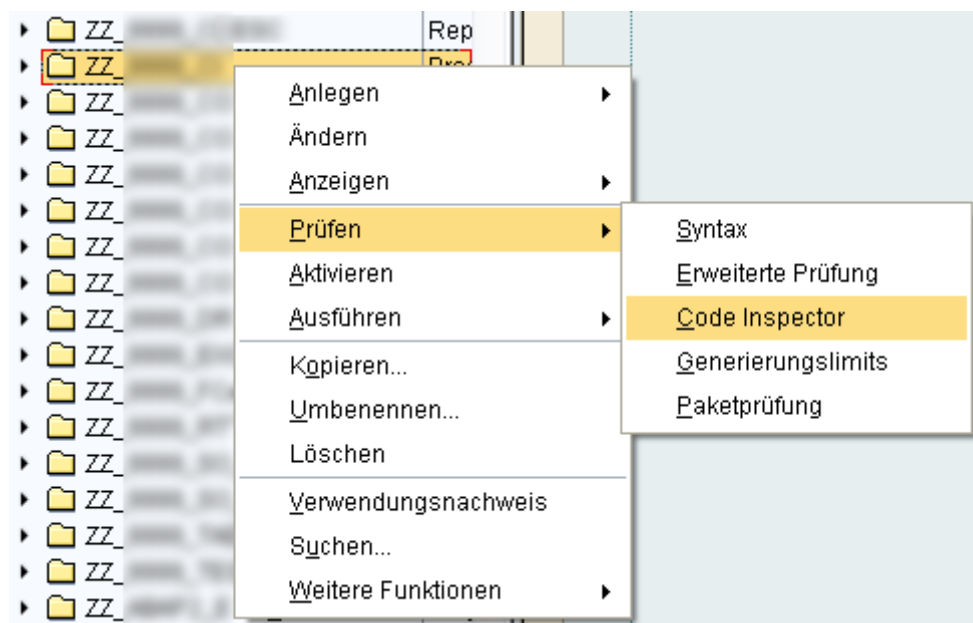


Abbildung 75: Aufruf des Code Inspectors: SAP-System-Screenshot

Dieser Aufruf führt zu einer Liste, in der ggf. die gefundenen Fehler aufgeführt werden.

Code Inspector: Ergebnisse							
Code Inspector							
Meldungen							
	D...	...	A...	Tests	Fehler	Warn...	Infor...
▾				Liste der Prüfungen	0	0	0
▸				Performance-Prüfungen	0	0	0
▸				Sicherheitsprüfungen	0	0	0
▸				Syntaxprüfung/Generierung	0	0	0
▸				Oberflächen	0	0	0

Abbildung 76: Ergebnisse des Code Inspectors: SAP-System-Screenshot

Durch Aufklappen des Baumes kann zu den einzelnen Fehlen navigiert werden. Ein Klick auf die **i**-Schaltfläche liefert nähere Informationen zu den Meldungen, etwa eine detaillierte Beschreibung oder einen Verbesserungsvorschlag. Weiterhin werden Hinweise angezeigt, wie die Meldung ausgeblendet werden kann. Hierzu dienen Pseudokommentare der Form

"#EC ...

Diese werden an das Ende der betroffenen Zeile gestellt. Der Grund hierfür ist, dass es in manchen Situationen nicht möglich oder sinnvoll ist, Code zu schreiben, der die Prüfungen besteht. Um in solchen Fällen nicht Unmengen von Fehlermeldungen zu erhalten, werden die Pseudokommentare eingesetzt.

Der Code Inspector differenziert den Umfang seiner Prüfungen durch sog. **Prüfvarianten**. Beim Aufruf aus dem ABAP Editor wird eine Default-Prüfvariante verwendet. Diese umfasst:

- eine Erweiterte Syntaxprüfung
- eine Prüfung auf kritische Anweisungen (etwa Native SQL)
- ausgewählte Performance-Prüfungen

Sie können eigene Prüfvarianten anlegen. Wenn Sie eine eigene Prüfvariante DEFAULT nennen, wird dadurch mandantenübergreifend für Ihren Benutzer die Default-Variante verschattet. Sie kann aber durch Löschen der benutzerdefinierten Variante wieder reaktiviert werden.

Zur Definition von Prüfvarianten müssen Sie den Code Inspector separat starten. Verwenden Sie hierzu die Transaktion **SCI**.

Hinweis: Um eine Transaktion in einem neuen Modus zu öffnen, gibt es das Präfix /o für das Kommandofeld (das Feld, in das Transaktionscodes eingegeben werden). Positionieren Sie den Cursor in dieses Feld und drücken Sie **F1**. Ihnen wird daraufhin die Hilfe für dieses Feld angezeigt. Machen Sie sich auch mit den weiteren Präfixes vertraut.

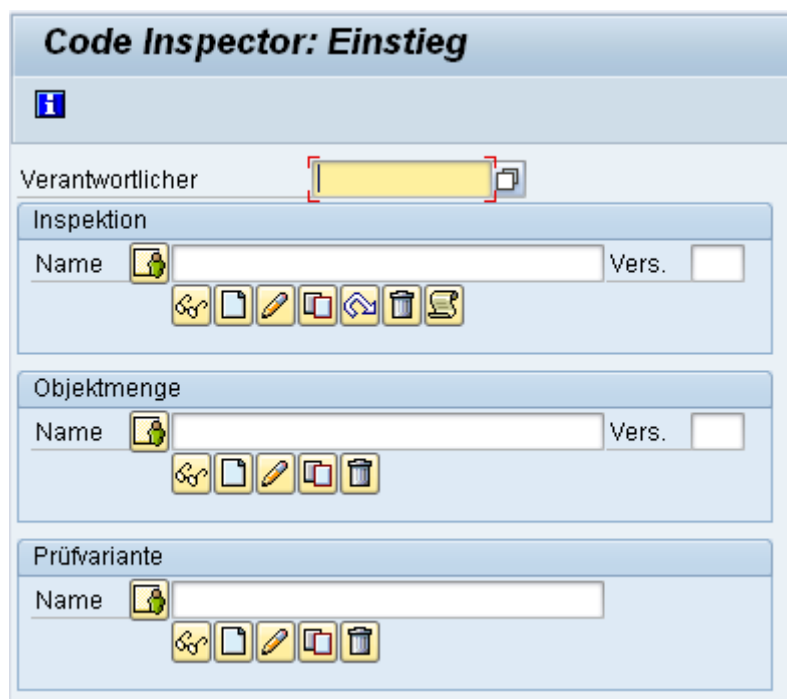


Abbildung 77: Einstiegsbild des Code Inspectors: SAP-System-Screenshot

Das Feld **Inspektion** dient der Benennung der eigentlichen Prüfung. Eine Angabe ist hier nur erforderlich, wenn das Ergebnis gespeichert werden oder die Ausführung im Hintergrund erfolgen soll. Fehlt die Angabe, wird das Ergebnis nicht gespeichert. Das Feld **Objektmenge** ermöglicht anzugeben, welche Repository-Objekte in die Prüfung einbezogen werden sollen. Im Feld **Prüfvariante** wird angegeben, welche Prüfungen durchgeführt werden sollen. Prüfvarianten und Objektmengen werden häufig vorher definiert, sie können aber auch temporär angelegt werden, so dass sie nur für eine Inspektion gelten und nicht wiederverwendet werden können. Alle drei Angaben können **lokal** oder **global** angelegt werden. Ersteres bewirkt einen Benutzerbezug, so dass nur eine Verwendung durch ihren Benutzer möglich ist, während globale Objekte systemweit allen Benutzern zur Verfügung stehen.

Die Prüfvariante besteht aus einer oder mehreren Prüfkategorien, die sich wiederum aus einer oder mehreren Einzelprüfungen zusammensetzen. Einzelprüfungen können Parameter besitzen, um etwa ein Kennzeichen für einen Teilaspekt der Prüfung angeben zu können.

Die wichtigsten Prüfkategorien sind:

- Allgemeine Prüfungen, die Datenaufbereitungen wie die Auflistung des Tabellennamens aus SELECT-Anfragen beinhalten
- Performance-Prüfungen, die Performance und Ressourcenverbrauch prüfen, etwa Analysen des WHERE-Teils von Datenbankabfragen oder unperformante Zugriffe auf interne Tabellen
- Syntaxprüfung/Generierung umfasst die Prüfung der ABAP-Syntax, eine erweiterte Programmprüfung und die Generierung
- Programmierkonventionen ermöglichen Prüfungen zu Namenskonventionen
- Suchfunktionen können zur Suche nach Wörtern im ABAP-Code verwendet werden

12.1 Praxis: Übung zum Code Inspector

Kopieren Sie das vordefinierte Programm **ZZ_CI** in Ihr Paket und geben Sie ihm den Namen **ZZ_####_CI**. Aktivieren Sie die Kopie und testen Sie sie. Machen Sie sich mit dem Code vertraut.

Führen Sie als nächstes eine erweiterte Programmprüfung durch. Wählen Sie hierzu aus dem Menü **Programm->Prüfen->Erweiterte Programmprüfung**. Bestätigen Sie die folgende Maske und schauen Sie sich die Ergebnisse an. Durch Doppelklick auf die Einträge können Sie sich Details anzeigen lassen:

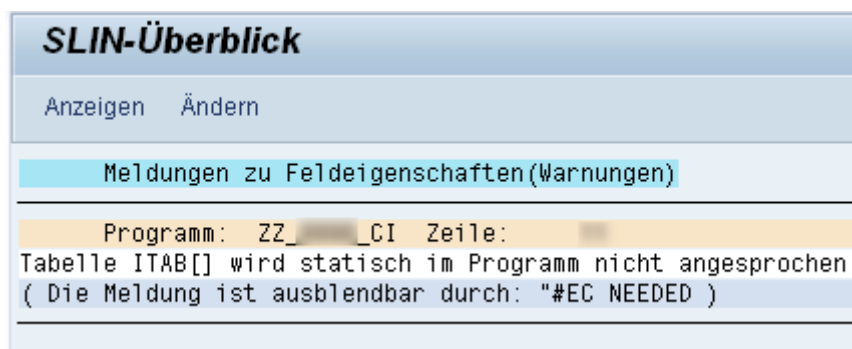



Abbildung 78: Details zu einer Warnung: SAP-System-Screenshot

Führen Sie als nächstes eine Prüfung mit der **Default-Prüfvariante** des Code Inspectors durch. Diese Prüfung können Sie aus dem Object Navigator anstoßen. In den Ergebnissen können andere Fehler erscheinen als bei der zuvor durchgeführten erweiterten Programmprüfung (oder auch gar keine).

Als nächstes werden Sie eine eigene Inspektion, Prüfvariante und Objektmenge verwenden. Starten Sie den Code Inspector dafür über die Transaktion **SCI**. Geben Sie als Name der Prüfvariante **ZZ_####_PV** an und klicken Sie auf . Selektieren Sie im Baum **Allgemeine Prüfungen, Performance-Prüfungen, Sicherheitsprüfungen** und **Syntaxprüfung/Generierung**. Öffnen Sie den Teilbaum **Sicherheitsprüfungen** und entfernen Sie den Haken bei **Suche nach bestimmten kritischen Anweisungen**. Öffnen Sie außerdem den Teilbaum **Syntaxprüfung/Generierung** und entfernen Sie dort den Haken bei **Syntaxprüfung in einem Remote-System**.




Wählen Sie außerdem den Unterpunkt **Suche von ABAP-Tokens** unterhalb von **Suchfunktionen** (klappen Sie den Baum dafür aus) und klicken Sie in dieser Zeile auf . Geben Sie an, dass Kommentare nach dem Suchstring ***TODO*** durchsucht werden sollen:

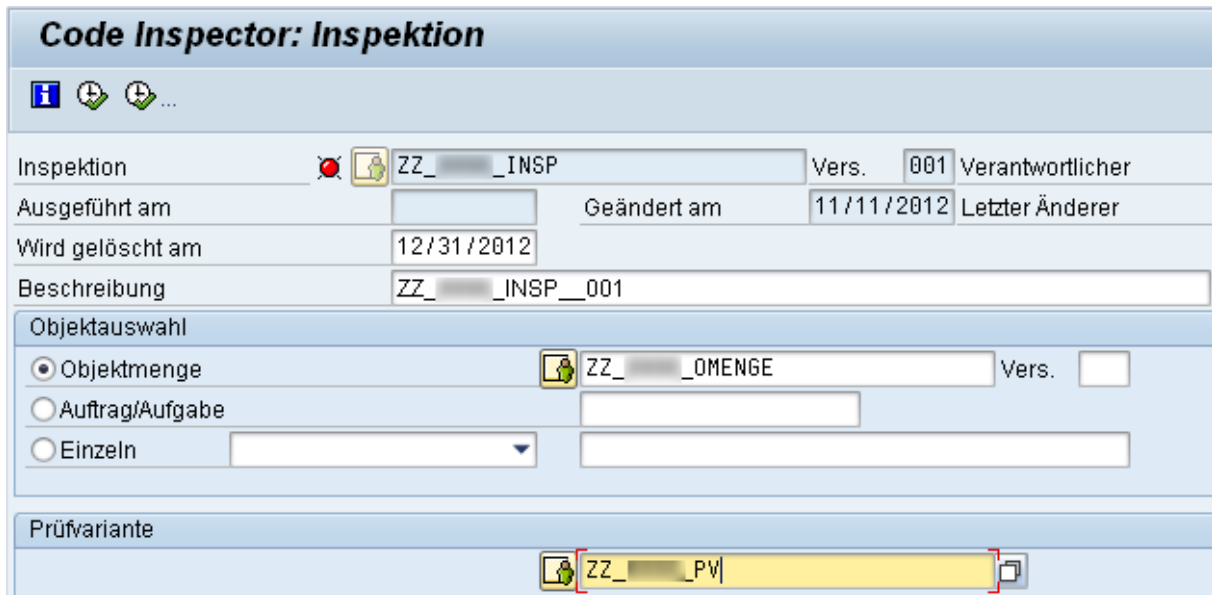


Abbildung 79: Konfiguration der Suche: SAP-System-Screenshot



Bestätigen Sie und wählen Sie im folgenden Fenster **Einzelwert** aus. Sichern Sie die Variante und kehren Sie zur Einstiegsmaske zurück.

Geben Sie als Namen für die Objektmenge **ZZ_####_OMENGE** an und legen Sie sie über  an. Geben Sie im unteren Bereich als Programm **ZZ_####_CI** an, und sorgen Sie dafür dass die Häkchen vor den anderen Einträgen in diesem Bereich nicht gesetzt sind. Sichern Sie und kehren Sie zur Einstiegsmaske zurück.

Vergeben Sie den Namen **ZZ_####_INSP** für die Inspektion und legen Sie diese ebenfalls über  an. Geben Sie für die Inspektion Ihre Prüfvariante und Ihre Objektmenge an:



Code Inspector: Inspektion


Inspektion   ZZ_..._INSP Vers. 001 Verantwortlicher

Ausgeführt am Geändert am 11/11/2012 Letzter Änderer

Wird gelöscht am 12/31/2012

Beschreibung ZZ_..._INSP_001

Objektauswahl

☒ Objektmenge  ZZ_..._OMENGE Vers.

☐ Auftrag/Aufgabe

☐ Einzel

Prüfvariante




 ZZ_..._PW 

Abbildung 80: Anlegen der Inspektion: SAP-System-Screenshot

Sichern Sie die Inspektion und führen Sie sie über  aus. Nach kurzer Zeit meldet das System, dass die Inspektion abgeschlossen wurde.

Wählen Sie den Menüpfad **Inspektion->Ergebnisse** und schauen Sie sich die Ergebnisse der Inspektion an.

Versuchen Sie anschließend sofern möglich die Probleme im Programm **ZZ_####_CI** zu beheben und wiederholen Sie die Prüfung. Wählen Sie hierzu in den Ergebnissen den Menüpfad **Hilfsmittel->Inspektion wiederholen** und Starten Sie die damit Inspektion erneut.

Es sollten nun **weniger Meldungen** erscheinen als zuvor.

Achtung: Sollten gar keine Meldungen erschienen sein, haben Sie in den schritten zuvor Fehler gemacht. Sie können dann entweder neu beginnen, oder sie führen die Korrekturen an den offensichtlichen Mängeln des Beispielprogramms „von Hand“ durch. Die Abgabe des Kapitels ohne Überarbeitung des **ZZ_####_CI**-Programms wird nicht als korrekte Lösung akzeptiert!

12.2Fazit: Code Inspector

Sie haben mit dem Code Inspector ein leistungsfähiges Werkzeug kennen gelernt, mit dem Sie umfangreiche Prüfungen von Entwicklungsobjekten durchführen können. Einer der Vorteile liegt darin, dass nicht nur ein einzelnes Programm geprüft werden kann, sondern durch die Objektmenge eine Auswahl zu prüfender Objekte möglich ist. Sollte eine Inspektion wegen ihres Umfangs zu lange dauern, können Sie diese auch im Hintergrund ausführen lassen.

12.3 Kapitelabschluss

Sie befinden sich am Ende dieses Abschnitts. Bevor sie die im folgenden Absatz beschriebene E-Mail verfassen, beachten Sie bitte die folgenden Hinweise:

1. Prüfen Sie, ob sie wirklich **alle** Aufgaben seit dem vorangegangenen Abschluss bearbeitet haben. Diese sind mit „Praxis:“ in der Überschrift gekennzeichnet (8.2, 8.4, 8.6, 8.8, 8.10, 10.3, 10.4, 10.5.2, 10.5.3, 10.6, 11.3, 12.1).
2. Prüfen Sie bitte noch einmal genau ob alle ihre Repository-Objekte **korrekt funktionieren**
3. Stellen Sie sicher, dass alle Repository-Objekte **aktiviert** sind. Um Objekte zu finden, die noch nicht aktiviert sind, wählen Sie aus dem Drop-Down-Menü oberhalb des Navigationsbaums im Object Navigator **Inaktive Objekte** aus. Geben Sie anschließend im darunter befindlichen Feld ihren Benutzernamen **USER#-###** ein und bestätigen Sie. Anschließend werden im Navigationsbaum die inaktiven Objekte dargestellt, die noch aktiviert werden müssen. Aktivieren Sie diese nun. Beachten Sie, dass sie die Zweige des Baums ggf. noch aufklappen müssen. Um zu ihrem Paket zurückzukehren, wählen Sie im Drop-Down-Menü wieder **Paket** aus und bestätigen Sie ihren Paketnamen.
4. Stellen Sie weiterhin sicher, dass die **Namen** ihrer Entwicklungsobjekte genau den Vorgaben im Skript entsprechen. Sollten Sie sich vertippt haben, können Sie Programme umbenennen, indem Sie diese mit der rechten Maustaste im Navigationsbaum des Object Navigators anklicken und **Umbenennen...** auswählen.

Diese Hinweise gelten auch für die folgenden Kapitel.

Wenn Sie den Kurs bis zu dieser Stelle bearbeitet haben, senden Sie bitte eine formlose E-Mail an die vom Kursbetreuer für diesen Kurs genannte Adresse mit dem Betreff „ABAP2: Abschluss Kapitel 8-12 User #####“ (die Anführungszeichen gehören nicht mit zum Betreff). Sie erhalten dann in Kürze Feedback (je nach Ergebnis **entweder** über den Fortschrittsbericht, wenn alles in Ordnung ist, **oder** per E-Mail, wenn noch Korrekturen nötig sind) und können Mängel ggf. noch nachbessern. Bitte achten Sie darauf, den Betreff genau wie angegeben zu formulieren, um eine effiziente Verarbeitung der Mail zu ermöglichen.

Sollten Sie Fragen haben, formulieren Sie diese bitte in einer **separaten E-Mail** mit aussagekräftigem Betreff, da die Kapitelabschlussmails meist nur über den Betreff verarbeitet werden!

13 Praxis: Fallstudie Universität

Zum Abschluss des Kurses haben Sie nun noch einmal in einem komplexen Beispiel die Gelegenheit, Ihre Fähigkeiten unter Beweis zu stellen. Die Aufgabenstellung ist bewusst offen gehalten, um Ihnen die Möglichkeit zu geben, sich selbständig für eine mögliche Lösungsvariante zu entscheiden.

13.1 Szenario

Die Hochschule Musterstadt führt in allen Bereichen der Hochschulverwaltung ein SAP-System ein, um eine einheitliche IT-Plattform für alle administrativen Aufgaben zu erhalten. Im Zuge dieser Einführung soll auch das Prüfungsamt der Hochschule zukünftig auf Basis von SAP arbeiten. Seitens der Mitarbeiter des Prüfungsamts, die bislang nur über wenig IT-Unterstützung verfügten, gibt es starke Vorbehalte gegen die Einführung. Aus diesem Grund soll im Rahmen eines Pilotprojekts zunächst eine einfache Verwaltung von Prüfungsergebnissen über SAP realisiert werden. In Folgeprojekten soll das System dann auf weitere Aufgabenbereiche wie die Verwaltung von Studiengängen, Prüfungsordnungen, Zeugnissen, Leistungsnachweisen etc. erweitert werden.

Für eine möglichst hohe Akzeptanz des Systems durch die Mitarbeiter sollen diese nicht mit einer gänzlich neuen Software konfrontiert werden. Es wird daher angestrebt, dass der zu entwickelnde Prototyp im Web-Browser läuft, der auf den Computern im Prüfungsamt bereits vorhanden ist und mit dessen Benutzung die Mitarbeiter vertraut sind. Der Projektleiter schlägt daher vor, eine Web Dynpro-Anwendung zu entwickeln, die die entsprechenden Aufgaben wahrnimmt, und beauftragt Sie mit der Umsetzung.

Bei der Anforderungserhebung ergibt sich, dass zunächst Studenten, Prüfungen und Prüfungsteilnahmen (Ergebnisse) verwaltet werden müssen. Zu den Studenten muss neben Name und Vorname eine Matrikelnummer und der Studiengang gespeichert werden. Zu Prüfungen muss eine ID, der Name und das Datum festgehalten werden, an dem die Prüfung stattgefunden hat, und ob es sich um eine mündliche oder schriftliche Prüfung handelt. Weiterhin sind der Prüfer und der Umfang der Prüfung (in Kreditpunkten) zu hinterlegen. Nimmt nun ein Student an einer Prüfung teil, ist diese Teilnahme mit der Prüfungs-ID und dem Ergebnis (Note) festzuhalten (wenn Sie möchten, können Sie auch eine Teilnahme-ID hinzufügen, das ist aber nicht erforderlich). Das Prüfungsamt möchte die Möglichkeit haben, zum einen alle Prüfungsergebnisse eines Studenten, zum anderen alle Ergebnisse einer bestimmten Prüfung anzeigen zu lassen. Selbstverständlich sollen Daten (Studenten, Prüfungen und Teilnahmen) neben dem anlegen auch geändert oder gelöscht werden können.

13.2 Aufgabe

Ihre Aufgabe ist es nun, anhand der Aussagen der Prüfungsamtsmitarbeiter eine Web Dynpro-Anwendung zu entwickeln. Sollten Ihnen Informationen fehlen, so treffen Sie an den entsprechenden Stellen geeignete Annahmen.

Nennen Sie Ihre Component bzw. Anwendung **ZZ_####_WD_PA**. Wählen Sie für die Datenbanktabellen passende Namen, die mit **ZZ####_** beginnen. Fangen Sie bei der

Entwicklung mit der Definition der Tabellen für Studenten, Prüfungen und Ergebnisse an, damit diese anschließend bei der Definition des Contexts zur Verfügung stehen.

Stellen Sie sicher, dass die Oberfläche Benutzerfreundlich ist. Insbesondere sollten fehlerhafte Eingaben abgefangen werden und geeignete Eingabehilfen für ID-Felder zur Verfügung stehen. Das heißt, dass Sie Suchhilfen definieren müssen, so dass nicht nur die IDs, sondern auch ein Name o.ä. (je nach Feld) zu sehen ist, anhand dessen der Benutzer den Wert auswählen kann.

Das Ändern und Anlegen von Datensätzen muss für den Benutzer klar **unterscheidbar** sein. Es muss für die Aktionen des Benutzers entsprechendes Feedback über Fehler- bzw. Bestätigungsmeldungen geben, z. B. eine Fehlermeldung beim Versuch, einen Datensatz mit einer bereits vorhandenen Nummer anzulegen oder einen Datensatz mit einer nicht vorhandenen Nummer zu öffnen.

Benutzen Sie bitte **keine Pflichtfelder** (außer an Stellen an denen fehlende Eingaben die Anwendung zu einem echten Absturz bringen würden). Es müssen sich auch Datensätze mit unvollständigen Angaben anlegen lassen. Diese Anforderung ist nicht aus dem Szenario begründet, sondern erleichtert uns das Testen ganz erheblich. Lösungen mit Pflichtfeldern müssen wir leider als fehlerhaft zurückweisen.

Bitte benutzen Sie für ihre Views **sinnvolle, sprechende Namen**. Dieser Punkt ist sehr wichtig, gerade wenn Sie Probleme haben und sich an den Tutor wenden. Wenn alle Views nur „view_1“, „view_2“, „student_7“ o.ä. heißen, wird der Tutor sich nicht in Ihrer Anwendung zurecht finden und sehr viel schlechter helfen können, als wenn die Views bspw. „student_anlegen“, „pruefung_anzeigen“ o.ä. heißen.

Ein Hinweis für besonders gewiefte Teilnehmer: Die geforderte Funktion, dass eine Ergebnisliste pro Student oder pro Prüfung angezeigt werden kann, muss natürlich ganz normal auf einer View implementiert werden – es ist nicht zulässig diese Funktion nur mithilfe einer komplexen Suchhilfe zu realisieren, in der diese Listen irgendwo auftauchen. Ebenfalls darf die Selektion von Datensätzen nicht allein über eine Suchhilfe implementiert werden (d.h. es muss möglich sein, ohne Verwendung einer Suchhilfe Datensätze zu laden).

Datenbankzugriffe können Sie am elegantesten in selbstdefinierten Methoden des Component Controllers kapseln. Diese Methoden können Sie dann aus den **onaction**-Methoden bzw. **wddommodify**-Methoden der Views aufrufen. Machen Sie Gebrauch vom Wizard, um bequem Eingabemasken und Zugriffe auf den Context zu generieren.

13.3 Kapitelabschluss

Sie befinden sich am Ende dieses Abschnitts. Bevor sie die im folgenden Absatz beschriebene E-Mail verfassen, beachten Sie bitte die folgenden Hinweise:

1. Prüfen Sie, ob sie wirklich **alle** Bestandteile des Fallstudienkapitels umgesetzt haben.
2. Prüfen Sie bitte noch einmal genau ob alle ihre Repository-Objekte **korrekt funktionieren**
3. Stellen Sie sicher, dass alle Repository-Objekte **aktiviert** sind. Um Objekte zu finden, die noch nicht aktiviert sind, wählen Sie aus dem Drop-Down-Menü oberhalb des Navigationsbaums im Object Navigator **Inaktive Objekte** aus. Geben Sie anschließend im darunter befindlichen Feld ihren Benutzernamen **USER#-###** ein und bestätigen Sie. Anschließend werden im Navigationsbaum die inaktiven Objekte dargestellt, die noch aktiviert werden müssen. Aktivieren Sie diese nun. Beachten Sie, dass sie die Zweige des Baums ggf. noch aufklappen müssen. Um zu ihrem Paket zurückzukehren, wählen Sie im Drop-Down-Menü wieder **Paket** aus und bestätigen Sie ihren Paketnamen.
4. Stellen Sie weiterhin sicher, dass die **Namen** ihrer Entwicklungsobjekte genau den Vorgaben im Skript entsprechen. Sollten Sie sich vertippt haben, können Sie Programme umbenennen, indem Sie diese mit der rechten Maustaste im Navigationsbaum des Object Navigators anklicken und **Umbenennen...** auswählen.

Wenn Sie den Kurs bis zu dieser Stelle bearbeitet haben, senden Sie bitte eine formlose E-Mail an die vom Kursbetreuer für diesen Kurs genannte Adresse mit dem Betreff „ABAP2: Abschluss Kapitel 13 User #####“ (die Anführungszeichen gehören nicht mit zum Betreff). Sie erhalten dann in Kürze Feedback (je nach Ergebnis **entweder** über den Fortschrittsbericht, wenn alles in Ordnung ist, **oder** per E-Mail, wenn noch Korrekturen nötig sind) und können Mängel ggf. noch nachbessern. Bitte achten Sie darauf, den Betreff genau wie angegeben zu formulieren, um eine effiziente Verarbeitung der Mail zu ermöglichen.

Sollten Sie Fragen haben, formulieren Sie diese bitte in einer **separaten E-Mail** mit aussagekräftigem Betreff, da die Kapitelabschlussmails meist nur über den Betreff verarbeitet werden!