Kostenloses eBook

LERNEN ABAP

Free unaffiliated eBook created from
Stack Overflow contributors.

#abap

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: abap

It is an unofficial and free ABAP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ABAP.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit ABAP

Bemerkungen

ABAP ist eine von SAP entwickelte Programmiersprache zur Programmierung von Geschäftsanwendungen im SAP-Umfeld.

Früher nur prozedural, ist ABAP dank der Erweiterung von ABAP Objects jetzt auch eine objektorientierte Sprache.

Versionen

Ausführung	Veröffentlichungsdatum
ABAP 7.50	2015-10-20
ABAP 7.40	2012-11-29
ABAP 7.0	2006-04-01
ABAP 6.40	2004-04-01
ABAP 6.20	2002-04-01
ABAP 6.10	2001-07-01
ABAP 4.6C	2001-04-01
ABAP 4.6A	1999-12-01
ABAP 4.5	1999-03-01
ABAP 4.0	1998-06-01
ABAP 3.0	1997-02-20

Examples

Hallo Welt

```
PROGRAM zhello_world.
START-OF-SELECTION.
    WRITE 'Hello, World!'.
```

Anstatt auf die Konsole zu drucken, schreibt ABAP Werte in eine Liste, die angezeigt wird, sobald die Hauptlogik ausgeführt wurde.

Hallo Welt in ABAP Objects

```
PROGRAM zhello_world.

CLASS main DEFINITION FINAL CREATE PRIVATE.
  PUBLIC SECTION.
    CLASS-METHODS: start.
ENDCLASS.

CLASS main IMPLEMENTATION.
  METHOD start.
    cl_demo_output=>display( 'Hello World!' ).
  ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
  main=>start( ).
```

Erste Schritte mit ABAP online lesen: https://riptutorial.com/de/abap/topic/1196/erste-schritte-mit-abap

Kapitel 2: ABAP GRID List Viewer (ALV)

Examples

ALV erstellen und anzeigen

In diesem Beispiel wird die einfachste ALV-Erstellung mit der Klasse `cl_salv_table` und ohne zusätzliche Formatierungsoptionen dargestellt. Zusätzliche Formatierungsoptionen werden nach dem `TRY ENDTRY` Block und vor dem `alv->display( )` Methode `alv->display( )` .

Alle nachfolgenden Beispiele, die den ABAP Objects-Ansatz für die ALV-Erstellung verwenden, verwenden dieses Beispiel als Ausgangspunkt.

```
DATA: t_spfli       TYPE STANDARD TABLE OF spfli,
      alv           TYPE REF TO cl_salv_table,
      error_message TYPE REF TO cx_salv_msg.

" Fill the internal table with example data
SELECT * FROM spfli INTO TABLE t_spfli.

" Fill ALV object with data from the internal table
TRY.
    cl_salv_table=>factory(
      IMPORTING
        r_salv_table = alv
      CHANGING
        t_table      = t_spfli ).
  CATCH cx_salv_msg INTO error_message.
    " error handling
ENDTRY.

" Use the ALV object's display method to show the ALV on the screen
alv->display( ).
```

Optimieren Sie die ALV-Spaltenbreite

In diesem Beispiel wird veranschaulicht, wie die Spaltenbreite optimiert wird, damit Spaltenüberschriften und Daten nicht abgeschnitten werden.

```
alv->get_columns( )->set_optimize( ).
```

Spalten in einem ALV ausblenden

In diesem Beispiel wird das `MANDT` Feld (Client) vor dem ALV `MANDT` . Beachten Sie, dass der an `get_column( )` Parameter `get_column( )` *muss* , damit dies funktioniert.

```
alv->get_columns( )->get_column( 'MANDT' )->set_visible( if_salv_c_bool_sap=>false ).
```

Umbenennen von Spaltenüberschriften in einem ALV

Der Spaltentext kann sich bei der horizontalen Größenänderung einer Spalte ändern. Dafür gibt es drei Methoden:

Methodenname	Maximale Länge der Überschrift
`set_short_text`	10
`set_medium_text`	20
`set_long_text`	40

Das folgende Beispiel zeigt die Verwendung aller drei. Ein `column` deklariert wird und als Hinweis auf das Ergebnis der instantiierten `alv->get_columns( )->get_column( 'DISTID' )` - `alv->get_columns( )->get_column( 'DISTID' )` . Der Spaltenname *muss aus* Großbuchstaben bestehen. Dies ist so, dass diese Methodenverkettung nur einmal in ihrer Instanziierung aufgerufen wird, anstatt jedes Mal ausgeführt zu werden, wenn eine Spaltenüberschrift geändert wird.

```
DATA column TYPE REF TO cl_salv_column.
column = alv->get_columns( )->get_column( 'DISTID' ).

column->set_short_text( 'Dist. Unit' ).
column->set_medium_text( 'Unit of Distance' ).
column->set_long_text( 'Mass Unit of Distance (kms, miles)' ).
```

Aktivieren Sie die ALV-Symbolleistenfunktionalität

Der folgende Methodenaufruf ermöglicht die Verwendung vieler erweiterter Funktionen wie Sortieren, Filtern und Exportieren von Daten.

```
alv->get_functions( )->set_all( ).
```

Alle anderen Zeilenstreifen in ALV aktivieren

Diese Methode erhöht die Lesbarkeit, indem aufeinanderfolgende Zeilen abwechselnd die Hintergrundfarbschattierung anzeigen.

```
alv->get_display_settings( )->set_striped_pattern( if_salv_c_bool_sap=>true ).
```

Titel eines angezeigten ALV einstellen

Wenn ein ALV angezeigt wird, ist der Titel ganz oben nur der Programmname. Mit dieser Methode kann der Benutzer einen Titel mit bis zu 70 Zeichen festlegen. Das folgende Beispiel zeigt, wie ein dynamischer Titel festgelegt werden kann, der die Anzahl der angezeigten Datensätze anzeigt.

```
alv->get_display_settings( )->set_list_header( |Flight Schedule - { lines( t_spfli ) }
records| ).
```

ABAP GRID List Viewer (ALV) online lesen: https://riptutorial.com/de/abap/topic/4660/abap-grid-list-viewer--alv-

Kapitel 3: ABAP-Objekte

Examples

Klassenerklärung

ABAP-Klassen können global oder lokal deklariert werden. Eine globale Klasse kann von jedem Objekt innerhalb des ABAP-Repositorys verwendet werden. Im Gegensatz dazu kann eine lokale Klasse nur innerhalb des angegebenen Bereichs verwendet werden.

```
CLASS lcl_abap_class DEFINITION.
  PUBLIC SECTION.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
ENDCLASS.
```

Konstruktor, Methoden

Klassenimplementierung:

```
CLASS lcl_abap_class DEFINITION.
  PUBLIC SECTION.
    METHODS: constructor,
             method1.
  PROTECTED SECTION.
  PRIVATE SECTION.
    METHODS: method2,
             method3.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD constructor.
        "Logic
    ENDMETHOD.

    METHOD method1.
        "Logic
    ENDMETHOD.

    METHOD method2.
        "Logic
        method3( ).
    ENDMETHOD.
```

```
    METHOD method3.
        "Logic
    ENDMETHOD.
ENDCLASS.
```

Beispiel für Methodenaufruf:

```
DATA lo_abap_class TYPE REF TO lcl_abap_class.
CREATE OBJECT lo_abap_class. "Constructor call
lo_abap_class->method1( ).
```

Methode mit Parametern (Importieren, Ändern, Exportieren)

Klassenimplementierung:

```
CLASS lcl_abap_class DEFINITION.
  PRIVATE SECTION.
    METHODS method1 IMPORTING iv_string TYPE string
                     CHANGING cv_string TYPE string
                    EXPORTING ev_string TYPE string.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD method1.
        cv_string = iv_string.
        ev_string = 'example'.
    ENDMETHOD.
ENDCLASS.
```

Beispiel für Methodenaufruf:

```
method1 (
  EXPORTING iv_string = lv_string
  IMPORTING ev_string = lv_string2
   CHANGING cv_string = lv_string3
).
```

Methode mit zurückkehrendem Parameter

Klassenimplementierung:

```
CLASS lcl_abap_class DEFINITION.
  PRIVATE SECTION.
    METHODS method1 RETURNING VALUE(rv_string) TYPE string.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD method1.
        rv_string = 'returned value'.
    ENDMETHOD.
ENDCLASS.
```

Beispiel für Methodenaufruf:

```
lv_string = method1( ).
```

Beachten Sie, dass mit `RETURNING` deklarierte Parameter nur als Wert übergeben werden.

Vererbung - Definition

Information

Mit Vererbung können Sie eine neue Klasse von einer vorhandenen Klasse ableiten. Dazu verwenden Sie den **Zusatz INHERITING FROM** im

CLASS Unterklasse **DEFINITION VON** Super **INHERITING.**

Aussage. Die neue Klassenunterklasse erbt alle Komponenten der vorhandenen Klassensuperklasse. Die neue Klasse wird als Unterklasse der Klasse bezeichnet, von der sie abgeleitet ist. Die ursprüngliche Klasse wird als Oberklasse der neuen Klasse bezeichnet. Eine Klasse kann mehr als eine direkte Unterklasse haben, jedoch nur eine direkte Oberklasse.

Klassenimplementierung

```
CLASS lcl_vehicle DEFINITION.
ENDCLASS.

CLASS lcl_vehicle IMPLEMENTATION.
ENDCLASS.

CLASS lcl_car DEFINITION INHERITING FROM lcl_vehicle.
ENDCLASS.

CLASS lcl_car IMPLEMENTATION.
ENDCLASS.
```

Vererbung - abstrakte und endgültige Methoden und Klassen

Information

Mit den **Ergänzungen ABSTRACT** und **FINAL** zu den **Anweisungen METHODS** und **CLASS** können Sie abstrakte und letzte Methoden oder Klassen definieren.

Eine abstrakte Methode ist in einer abstrakten Klasse definiert und kann nicht in dieser Klasse implementiert werden. Stattdessen wird es in einer Unterklasse der Klasse implementiert. Abstrakte Klassen können nicht instanziiert werden.

Eine letzte Methode kann in einer Unterklasse nicht neu definiert werden. Abschlussklassen können keine Unterklassen haben. Sie schließen einen Vererbungsbaum ab.

Klassenimplementierung:

```
CLASS lcl_abstract DEFINITION ABSTRACT.
    PUBLIC SECTION.
        METHODS: abstract_method ABSTRACT,
                 final_method FINAL
                 normal_method.

ENDCLASS.

CLASS lcl_abstract IMPLEMENTATION.
    METHOD final_method.
        "This method can't be redefined in child class!
    ENDMETHOD.

    METHOD normal_method.
        "Some logic
    ENDMETHOD.

        "We can't implement abstract_method here!

ENDCLASS.

CLASS lcl_abap_class DEFINITION INHERITING FROM lcl_abstract.
    PUBLIC SECTION.
        METHODS: abstract_method REDEFINITION,
                 abap_class_method.
ENDCLASS.

CLASS lcl_abap_class IMPLEMENTATION.
    METHOD abstract_method.
        "Abstract method implementation
    ENDMETHOD.

    METHOD abap_class_method.
        "Logic
    ENDMETHOD.
ENDCLASS.
```

Beispiel für Methodenaufruf:

```
DATA lo_class TYPE REF TO lcl_abap_class.
CREATE OBJECT lo_class.

lo_class->abstract_method( ).
lo_class->normal_method( ).
lo_class->abap_class_method( ).
lo_class->final_method( ).
```

ABAP-Objekte online lesen: https://riptutorial.com/de/abap/topic/2244/abap-objekte

Kapitel 4: Bemerkungen

Examples

Ende der Linie

Jeder Text, der auf ein " Zeichen " in derselben Zeile folgt, wird auskommentiert:

```
DATA ls_booking TYPE flightb. " Commented text
```

Volle Linie

Das Zeichen * kommentiert eine ganze Zeile aus. Das * muss das erste Zeichen in der Zeile sein.

```
* DATA ls_booking TYPE flightb. Nothing on this line will be executed.
```

Bemerkungen online lesen: https://riptutorial.com/de/abap/topic/1644/bemerkungen

Kapitel 5: Datenerklärung

Examples

Inline-Datendeklaration

In bestimmten Situationen können Datendeklarationen inline durchgeführt werden.

```
LOOP AT lt_sflight INTO DATA(ls_sflight).
    WRITE ls_sflight-carrid.
ENDLOOP.
```

Einzelvariablendeklaration

```
DATA begda TYPE sy-datum.
```

Deklaration mehrerer Variablen

```
DATA: begda TYPE sy-datum,
      endda TYPE sy-datum.
```

Inline-Datendeklaration in SELECT-Anweisung

Bei Verwendung einer Inline-Datendeklaration innerhalb eines `SELECT...ENDSELECT` Blocks oder einer `SELECT SINGLE` Anweisung muss das `@` -Zeichen als Escape-Zeichen für den Ausdruck `DATA(lv_cityto)` verwendet werden. Nachdem das Escape-Zeichen verwendet wurde, müssen auch alle weiteren Host-Variablen mit `lv_carrid` werden (wie im Fall von `lv_carrid` unten).

```
DATA lv_carrid TYPE s_carr_id VALUE 'LH'.
SELECT SINGLE cityto FROM spfli
       INTO @DATA(lv_cityto)
       WHERE carrid = @lv_carrid
       AND   connid = 2402.
WRITE: / lv_cityto.
```

Ausgänge `BERLIN` .

Variablendeklarationsoptionen

Verschiedene Arten von Variablen können mit speziellen Optionen deklariert werden.

```
DATA: lv_string   TYPE string, " standard declaration
      lv_char     TYPE c,      " declares a character variable of length 1
      lv_char5(5) TYPE c,      " declares a character variable of length 5
      l_packed TYPE p LENGTH 10 DECIMALS 5 VALUE '1234567890.123456789'. " evaluates to
1,234,567,890.12346
```

Datenerklärung online lesen: https://riptutorial.com/de/abap/topic/1646/datenerklarung

Kapitel 6: Dynamische Programmierung

Examples

Feldsymbole

Feldsymbole sind ABAPs Äquivalent zu Zeigern, außer dass Feldsymbole immer dereferenziert werden (es ist nicht möglich, die tatsächliche Adresse im Speicher zu ändern).

Erklärung

Um ein `FIELD-SYMBOLS` zu `FIELD-SYMBOLS` muss das Schlüsselwort `FIELD-SYMBOLS` verwendet werden. Typen können generisch sein (`ANY [... TABLE]`), um eine Vielzahl von Variablen zu verarbeiten.

```
FIELD-SYMBOLS: <fs_line>     TYPE any,    "generic
               <fs_struct>   TYPE kna1.   "non-generic
```

Zuweisung

Feldsymbole werden bei der Deklaration `unassigned` , `unassigned` sie zeigen auf nichts. Der Zugriff auf ein nicht zugewiesenes Feldsymbol führt zu einer Ausnahme und, falls nicht abgerufen, zu einem kurzen Speicherauszug. Daher sollte der Zustand mit `IS ASSIGNED` überprüft werden:

```
IF <fs> IS ASSIGNED.
*... symbol is assigned
ENDIF.
```

Da es sich nur um Referenzen handelt, können keine echten Daten darin gespeichert werden. Daher sind deklarierte `DATA` in jedem Anwendungsfall erforderlich.

```
DATA: w_name  TYPE string VALUE `Max`,
      w_index TYPE i      VALUE 1.

FIELD-SYMBOLS <fs_name> TYPE any.

ASSIGN w_name TO <fs_name>. "<fs_name> now gets w_name
<fs_name> = 'Manni'.        "Changes to <fs_name> now also affect w_name

* As <fs_name> is generic, it can also be used for numbers

ASSIGN w_index TO <fs_name>. "<fs_name> now refers to w_index.
ADD 1 TO <fs_name>.          "w_index gets incremented by one
```

Zuordnung aufheben

Manchmal kann es nützlich sein, ein Feldsymbol zurückzusetzen. Dies kann mit `UNASSIGN` .

```
UNASSIGN <fs>.
* Access on <fs> now leads to an exception again
```

Verwenden Sie für interne Tabellen

Feldsymbole können verwendet werden, um interne Tabellen zu ändern.

```
LOOP AT itab INTO DATA(wa).
* Only modifies wa_line
    wa-name1 = 'Max'.
ENDLOOP.

LOOP AT itab ASSIGNING FIELD-SYMBOL(<fs>).
* Directly refers to a line of itab and modifies its values
    <fs>-name1 = 'Max'.
ENDLOOP.
```

Beachtung! Feldsymbole bleiben auch nach Verlassen der Schleife zugewiesen. Wenn Sie sie sicher wiederverwenden möchten, heben Sie die Zuweisung sofort auf.

Datenreferenzen

Für Datenreferenzen ist der Zusatz `REF TO` nach `TYPE` .

Dynamische Erstellung von Strukturen

Wenn der Typ einer Struktur zur Laufzeit festgelegt werden soll, können wir unsere Zielstruktur als Referenz auf die generischen `data` .

```
DATA wa TYPE REF TO data.
```

Um `wa` einen Typ zu geben, verwenden wir die Anweisung `CREATE DATA` . Der Zusatz `TYPE` kann angegeben werden durch:

Referenz:

```
CREATE DATA wa TYPE kna1
```

- *Statische Prüfungen sind aktiv, daher ist es nicht möglich, einen unbekannten Typ zu erstellen*

Name:

```
CREATE DATA wa TYPE (lw_name_as_string)
```

- *Die Klammern werden benötigt und `lw_name_as_string` enthält den `lw_name_as_string` als String.*
- *Wenn der Typ nicht gefunden wurde, wird eine Ausnahme vom Typ `CX_SY_CREATE_DATA_ERROR` ausgelöst*

Für das Instanziieren von dynamisch erstellten Typen kann der Zusatz `HANDLE` angegeben werden. `HANDLE` erhält ein Objekt, das von `CL_ABAP_DATADESCR` erbt.

```
CREATE DATA dref TYPE HANDLE obj
```

- `obj`

kann mit dem **R** un **T** ime **T** yp **S** ervice erstellt werden
- da `dref` noch eine Datenreferenz ist, muss es dereferenziert werden (`->*`), um als Datencontainer verwendet zu werden (normalerweise über Feldsymbole)

RunTime Type Services

RunTime Type Services (*kurz:* **RTTS**) werden verwendet für:

- Typen erstellen (RunTime Type Creation; *kurz:* **RTTC**)
- Typen analysieren (RunTime Type Identification; *kurz:* **RTTI**)

Klassen

```
CL_ABAP_TYPEDESCR
  |
  |--CL_ABAP_DATADESCR
  |    |
  |    |--CL_ABAP_ELEMDESCR
  |    |--CL_ABAP_REFDESCR
  |    |--CL_ABAP_COMPLEXDESCR
  |         |
  |         |--CL_ABAP_STRUCTDESCR
  |         |--CL_ABAP_TABLEDESCR
  |
  |--CL_ABAP_OBJECTDESCR
     |
     |--CL_ABAP_CLASSDESCR
     |--CL_ABAP_INTFDESCR
```

`CL_ABAP_TYPEDESCR` ist die Basisklasse. Es implementiert die erforderlichen Methoden zur Beschreibung:

- `DESCRIBE_BY_DATA`
- `DESCRIBE_BY_NAME`
- `DESCRIBE_BY_OBJECT_REF`
- `DESCRIBE_BY_DATA_REF`

Dynamische Programmierung online lesen: https://riptutorial.com/de/abap/topic/4442/dynamische-programmierung

Kapitel 7: Interne Tabellen

Examples

Arten von internen Tabellen

```
DATA: <TABLE NAME> TYPE <SORTED|STANDARD|HASHED> TABLE OF <TYPE NAME>
      WITH <UNIQUE|NON-UNIQUE> KEY <FIELDS FOR KEY>.
```

Standardtabelle

In dieser Tabelle werden alle Einträge linear gespeichert, und auf Datensätze wird linear zugegriffen. Bei großen Tabellengrößen kann der Tabellenzugriff langsam sein.

Sortierte Tabelle

Erfordert den Zusatz `WITH UNIQUE | NON-UNIQUE KEY` . Die Suche ist aufgrund der binären Suche schnell. Einträge können nicht an diese Tabelle angehängt werden, da dies die Sortierreihenfolge `INSERT` . Sie werden daher immer mit dem Schlüsselwort `INSERT` eingefügt.

Hash-Tabelle

Erfordert den Zusatz `WITH UNIQUE | NON-UNIQUE KEY` . Verwendet einen proprietären Hash-Algorithmus, um Schlüsselwertpaare zu erhalten. Theoretisch kann die Suche so langsam sein wie die `STANDARD` Tabelle, aber praktisch sind sie schneller als eine `SORTED` Tabelle, die unabhängig von der Größe der Tabelle eine konstante Zeit benötigt.

Deklaration interner ABAP-Tabellen

Interne Tabellendeklaration basierend auf lokaler Typdefinition

```
" Declaration of type
TYPES: BEGIN OF ty_flightb,
         id        TYPE fl_id,
         dat       TYPE fl_date,
         seatno    TYPE fl_seatno,
         firstname TYPE fl_fname,
         lastname  TYPE fl_lname,
         fl_smoke  TYPE fl_smoker,
         classf    TYPE fl_class,
         classb    TYPE fl_class,
         classe    TYPE fl_class,
         meal      TYPE fl_meal,
         service   TYPE fl_service,
         discout   TYPE fl_discnt,
       END OF lty_flightb.
```

```
" Declaration of internal table
DATA t_flightb TYPE STANDARD TABLE OF ty_flightb.
```

Deklaration basierend auf Datenbanktabelle

```
DATA t_flightb TYPE STANDARD TABLE OF flightb.
```

Interne Deklaration der Inline-Tabelle

Erfordert ABAP-Version> 7.4

```
TYPES t_itab TYPE STANDARD TABLE OF i WITH EMPTY KEY.

DATA(t_inline) = VALUE t_itab( ( 1 ) ( 2 ) ( 3 ) ).
```

Interne Tabelle mit Headerzeilen-Deklaration

In ABAP gibt es Tabellen mit Kopfzeilen und Tabellen ohne Kopfzeilen. Tabellen mit Kopfzeilen sind ein älteres Konzept und sollten bei Neuentwicklungen nicht verwendet werden.

Interne Tabelle: Standardtabelle mit / ohne Kopfzeile

Dieser Code deklariert die Tabelle `i_compc_all` mit der vorhandenen Struktur von `compc_str` .

```
DATA: i_compc_all TYPE STANDARD TABLE OF compc_str WITH HEADER LINE.
DATA: i_compc_all TYPE STANDARD TABLE OF compc_str.
```

Interne Tabelle: Hash-Tabelle mit / ohne Kopfzeile

```
DATA: i_map_rules_c TYPE HASHED TABLE OF /bic/ansdomm0100 WITH HEADER LINE
DATA: i_map_rules_c TYPE HASHED TABLE OF /bic/ansdomm0100
```

Deklaration eines Arbeitsbereichs für Tabellen ohne Kopf

Ein Arbeitsbereich (im Allgemeinen als *wa* abgekürzt) hat dieselbe Struktur wie die Tabelle, kann jedoch nur eine Zeile enthalten (ein WA ist eine Struktur einer Tabelle mit nur einer Dimension).

```
DATA: i_compc_all_line LIKE LINE OF i_compc_all.
```

Lesen, Schreiben und Einfügen in interne Tabellen

Lesen, Schreiben und Einfügen in interne Tabellen mit Kopfzeile:

```
" Read from table with header (using a loop):
LOOP AT i_compc_all.              " Loop over table i_compc_all and assign header line
  CASE i_compc_all-ftype.         " Read cell ftype from header line from table i_compc_all
    WHEN 'B'.                     " Bill-to customer number transformation
      i_compc_bil = i_compc_all.  " Assign header line of table i_compc_bil with content of
header line i_compc_all
      APPEND i_compc_bil.         " Insert header line of table i_compc_bil into table
i_compc_bil
    " ... more WHENs
  ENDCASE.
ENDLOOP.
```

> Zur Erinnerung: Interne Tabellen mit Kopfzeilen sind in objektorientierten Kontexten verboten. Die Verwendung interner Tabellen *ohne* Kopfzeilen wird immer empfohlen.

Lesen, Schreiben und Einfügen in interne Tabellen ohne Kopfzeile:

```
" Loop over table i_compc_all and assign current line to structure i_compc_all_line
LOOP AT i_compc_all INTO i_compc_all_line.
  CASE i_compc_all_line-ftype.                " Read column ftype from current line (which as
assigned into i_compc_all_line)
    WHEN 'B'.                                 " Bill-to customer number transformation
      i_compc_bil_line = i_compc_all_line.    " Copy structure
      APPEND i_compc_bil_line TO i_compc_bil. " Append structure to table
    " more WHENs ...
  ENDCASE.
ENDLOOP.


" Insert into table with Header:
INSERT TABLE i_sap_knb1.                      " insert into TABLE WITH HEADER: insert table
header into it's content
insert i_sap_knb1_line into table i_sap_knb1. " insert into HASHED TABLE: insert structure
i_sap_knb1_line into hashed table i_sap_knb1
APPEND p_t_errorlog_line to p_t_errorlog.     " insert into STANDARD TABLE: insert structure /
wa p_t_errorlog_line into table p_t_errorlog_line
```

Interne Tabellen online lesen: https://riptutorial.com/de/abap/topic/1647/interne-tabellen

Kapitel 8: Nachrichtenklassen / MESSAGE-Schlüsselwort

Einführung

Die `MESSAGE` Anweisung kann verwendet werden, um den Programmablauf zu unterbrechen, um dem Benutzer Kurznachrichten anzuzeigen. Der Inhalt der Nachrichten kann im Programmcode, in den `SE91` des Programms oder in einer in `SE91` definierten unabhängigen Nachrichtenklasse definiert `SE91` .

Bemerkungen

Die maximale Länge einer Nachricht, einschließlich der mit `&` übermittelten Parameter, beträgt 72 Zeichen.

Examples

Definieren einer Nachrichtenklasse

```
PROGRAM zprogram MESSAGE-ID sabapdemos.
```

Systemdefinierte Nachrichten können in einer Nachrichtenklasse gespeichert werden. Das `MESSAGE-ID` Token definiert die Nachrichtenklasse `sabapdemos` für das gesamte Programm. Wenn dies nicht verwendet wird, muss die Nachrichtenklasse bei jedem `MESSAGE` Aufruf angegeben werden.

NACHRICHT mit vordefiniertem Textsymbol

```
PROGRAM zprogram MESSAGE-ID za.
...
MESSAGE i000 WITH TEXT-i00.
```

In einer Nachricht wird dem Benutzer der im `i00` gespeicherte Text `i00` . Da der Nachrichtentyp i ist (wie in `i000`), wird der Programmablauf ab dem Punkt des `MESSAGE` Aufrufs fortgesetzt, nachdem der Benutzer das Dialogfeld verlassen hat.

Obwohl der Text nicht aus der Nachrichtenklasse `za` , muss eine `MESSAGE-ID` angegeben werden.

Nachricht ohne vordefinierte Nachrichtenklasse

```
PROGRAM zprogram.
...
MESSAGE i050(sabapdemos).
```

Es kann unbequem sein, eine Nachrichtenklasse für das gesamte Programm zu definieren. `MESSAGE` ist es möglich, die Nachrichtenklasse, aus der die Nachricht stammt, in der `MESSAGE` Anweisung selbst zu definieren. In diesem Beispiel wird die Nachricht `050` der Nachrichtenklasse `sabapdemos` .

Dynamisches Messaging

```
DATA: msgid TYPE sy-msgid VALUE 'SABAPDEMOS',
      msgty TYPE sy-msgty VALUE 'I',
      msgno TYPE sy-msgno VALUE '050'.

MESSAGE ID mid TYPE mtype NUMBER num.
```

Der `MESSAGE` Aufruf oben ist auch der Anruf `MESSAGE i050(sapdemos). .`

Parameter an Nachrichten übergeben

Das `&` -Symbol kann in einer Nachricht verwendet werden, damit Parameter übergeben werden können.

Geordnete Parameter

Nachricht `777` der Klasse `sabapdemos` :

```
Message with type &1 &2 in event &3
```

Wenn Sie diese Nachricht mit drei Parametern aufrufen, wird eine Nachricht mit den Parametern zurückgegeben:

```
MESSAGE i050(sabapdemos) WITH 'E' '010' 'START-OF-SELECTION`.
```

Diese Nachricht wird `Message with type E 010 in event START-OF-SELECTION` als `Message with type E 010 in event START-OF-SELECTION` angezeigt. Die Zahl neben dem `&` -Symbol kennzeichnet die Reihenfolge, in der die Parameter angezeigt werden.

Ungeordnete Parameter

Nachricht `888` der Klasse `sabapdemos` :

```
& & & &
```

Der Aufruf dieser Nachricht ist ähnlich:

```
MESSAGE i050(sabapdemos) WITH 'param1' 'param2' 'param3' 'param4'.
```

Dies gibt `param1 param2 param3 param4` .

Nachrichtenklassen / MESSAGE-Schlüsselwort online lesen:
https://riptutorial.com/de/abap/topic/10691/nachrichtenklassen---message-schlusselwort

Kapitel 9: Öffnen Sie SQL

Examples

SELECT-Anweisung

SELECT ist eine Open-SQL-Anweisung zum Lesen von Daten aus einer oder mehreren Datenbanktabellen in Datenobjekte .

1. Alle Datensätze auswählen

```
* This returns all records into internal table lt_mara.
SELECT * FROM mara
         INTO lt_mara.
```

2. Einzelnen Datensatz auswählen

```
* This returns single record if table consists multiple records with same key.
SELECT SINGLE * INTO TABLE lt_mara
                FROM mara
                WHERE matnr EQ '400-500'.
```

3. Auswahl einzelner Datensätze

```
* This returns records with distinct values.
SELECT DISTINCT * FROM mara
                  INTO TABLE lt_mara
                  ORDER BY matnr.
```

4. Aggregatfunktionen

```
* This puts the number of records present in table MARA into the variable lv_var
SELECT COUNT( * ) FROM mara
                  INTO lv_var.
```

Öffnen Sie SQL online lesen: https://riptutorial.com/de/abap/topic/6885/offnen-sie-sql

Kapitel 10: Regeln der Namensgebung

Syntax

- Zeichen, Zahlen und _ können für den Variablennamen verwendet werden.
- Zwei Zeichen für Variablenstatus und Objekttyp.
- Lokale Variablen beginnen mit L.
- Globale Variablen beginnen mit G.
- Funktionseingangsparameter beginnen mit I (Import).
- Funktionsausgabeparameter beginnen mit E (Export).
- Struktursymbol ist S.
- Tabellensymbol ist T.

Examples

Lokale Variable

```
data: lv_temp type string.
data: ls_temp type sy.
data: lt_temp type table of sy.
```

Globale Variable

```
data: gv_temp type string.
data: gs_temp type sy.
data: gt_temp type table of sy.
```

Regeln der Namensgebung online lesen: https://riptutorial.com/de/abap/topic/6770/regeln-der-namensgebung

Kapitel 11: Reguläre Ausdrücke

Examples

Ersetzen

Die `REPLACE` Anweisung kann direkt mit regulären Ausdrücken arbeiten:

```
DATA(lv_test) = 'The quick brown fox'.
REPLACE ALL OCCURRENCES OF REGEX '\wo' IN lv_test WITH 'XX'.
```

Die Variable `lv_test` wird zu `The quick bXXwn XXx` .

Suchen

Die Anweisung `FIND` kann direkt mit regulären Ausdrücken arbeiten:

```
DATA(lv_test) = 'The quick brown fox'.

FIND REGEX '..ck' IN lv_test.
" sy-subrc == 0

FIND REGEX 'a[sdf]g' IN lv_test.
" sy-subrc == 4
```

Objektorientierte reguläre Ausdrücke

Für fortgeschrittenere `CL_ABAP_REGEX` Sie am besten `CL_ABAP_REGEX` und die zugehörigen Klassen.

```
DATA: lv_test  TYPE string,
      lo_regex TYPE REF TO cl_abap_regex.

lv_test = 'The quick brown fox'.
CREATE OBJECT lo_regex
  EXPORTING
    pattern = 'q(...)\w'.

DATA(lo_matcher) = lo_regex->create_matcher( text = lv_test ).
WRITE: / lo_matcher->find_next( ).      " X
WRITE: / lo_matcher->get_submatch( 1 ). " uic
WRITE: / lo_matcher->get_offset( ).     " 4
```

Auswerten regulärer Ausdrücke mit einer Prädikatsfunktion

Die `matches` Prädikatsfunktion kann verwendet werden, um Zeichenfolgen ohne Objektdeklarationen im laufenden Betrieb auszuwerten.

```
IF matches( val = 'Not a hex string'
            regex = '[0-9a-f]*' ).
```

```
  cl_demo_output=>display( 'This will not display' ).
ELSEIF matches( val = '6c6f7665'
                regex = '[0-9a-f]*' ).
  cl_demo_output=>display( 'This will display' ).
ENDIF.
```

SubMatches mit OO-regulären Ausdrücken abrufen

Mit der Methode `GET_SUBMATCH` der Klasse `CL_ABAP_MATCHER` können wir die Daten in den Gruppen / Untergruppen `CL_ABAP_MATCHER` .

Ziel: Holen Sie sich das Token rechts neben dem Schlüsselwort 'Type'.

```
DATA: lv_pattern TYPE string VALUE 'type\s+(\w+)',
      lv_test TYPE string VALUE 'data lwa type mara'.

CREATE OBJECT ref_regex
  EXPORTING
        pattern     = lv_pattern
        ignore_case = c_true.

ref_regex->create_matcher(
    EXPORTING
        text    = lv_test
    RECEIVING
        matcher = ref_matcher
       ).

ref_matcher->get_submatch(
            EXPORTING
                 index = 0
            RECEIVING
                 submatch = lv_smatch.
```

Die resultierende Variable `lv_smatch` enthält den Wert `MARA` .

Reguläre Ausdrücke online lesen: https://riptutorial.com/de/abap/topic/5113/regulare-ausdrucke

Kapitel 12: Schleifen

Bemerkungen

Beim Durchlaufen von internen Tabellen ist es im Allgemeinen vorzuziehen, einem `ASSIGN` , anstatt `INTO` einen Arbeitsbereich zu schleifen. Durch das Zuweisen von Feldsymbolen wird der Verweis auf die nächste Zeile der internen Tabelle bei jeder Iteration aktualisiert. Die Verwendung von `INTO` führt dazu, dass die Zeile der Tabelle in den Arbeitsbereich kopiert wird, was für lange / breite Tabellen teuer sein kann.

Examples

Interne Tabellenschleife

```
LOOP AT itab INTO wa.
ENDLOOP.

FIELD-SYMBOLS <fs> LIKE LINE OF itab.
LOOP AT itab ASSIGNING <fs>.
ENDLOOP.

LOOP AT itab ASSIGNING FIELD-SYMBOL(<fs>).
ENDLOOP.

LOOP AT itab REFERENCE INTO dref.
ENDLOOP.

LOOP AT itab TRANSPORTING NO FIELDS.
ENDLOOP.
```

Bedingtes Looping

Wenn nur Zeilen mit einer bestimmten Bedingung in die Schleife aufgenommen werden sollen, kann der Zusatz `WHERE` hinzugefügt werden.

```
LOOP AT itab INTO wa WHERE f1 = 'Max'.
ENDLOOP.
```

While-Schleife

ABAP bietet auch das herkömmliche `WHILE` -Loop an, das ausgeführt wird, bis der angegebene Ausdruck als falsch ausgewertet wird. Das Systemfeld `sy-index` wird für jeden Loop-Schritt erhöht.

```
WHILE condition.
* do something
ENDWHILE
```

Machen Sie eine Schleife

Ohne Zusatz läuft der `DO` -Loop endlos oder zumindest solange, bis er explizit von innen heraus verlassen wird. Das Systemfeld `sy-index` wird für jeden Loop-Schritt erhöht.

```
DO.
* do something... get it?
* call EXIT somewhere
ENDDO.
```

Der Zusatz `TIMES` bietet eine sehr bequeme Möglichkeit, den Code zu wiederholen (`amount` entspricht einem Wert vom Typ `i`).

```
DO amount TIMES.
* do several times
ENDDO.
```

Allgemeine Befehle

Um die Schleifen zu brechen, kann der Befehl `EXIT` verwendet werden.

```
DO.
    READ TABLE itab INDEX sy-index INTO DATA(wa).
    IF sy-subrc <> 0.
        EXIT. "Stop this loop if no element was found
    ENDIF.
    " some code
ENDDO.
```

Um zum nächsten Loop-Schritt zu `CONTINUE` kann der Befehl `CONTINUE` verwendet werden.

```
DO.
    IF sy-index MOD 1 = 0.
        CONTINUE. " continue to next even index
    ENDIF.
    " some code
ENDDO.
```

Die `CHECK` Anweisung ist eine `CONTINUE` Bedingung. Wenn sich die Bedingung als **falsch** `CONTINUE` wird `CONTINUE` ausgeführt. Mit anderen Worten: *Die Schleife führt den Schritt nur dann weiter, wenn die Bedingung erfüllt ist* .

Dieses Beispiel von `CHECK` ...

```
DO.
    " some code
    CHECK sy-index < 10.
    " some code
ENDDO.
```

... ist äquivalent zu ...

```
DO.
```

```
    " some code
    IF sy-index >= 10.
        CONTINUE.
    ENDIF.
    " some code
ENDDO.
```

Schleifen online lesen: https://riptutorial.com/de/abap/topic/2270/schleifen

Kapitel 13: Steuerungsflussanweisungen

Examples

IF / ELSEIF / ELSE

```
IF lv_foo = 3.
  WRITE: / 'lv_foo is 3'.
ELSEIF lv_foo = 5.
  WRITE: / 'lv_foo is 5'.
ELSE.
  WRITE: / 'lv_foo is neither 3 nor 5'.
ENDIF.
```

FALL

```
CASE lv_foo.
  WHEN 1.
    WRITE: / 'lv_foo is 1'.
  WHEN 2.
    WRITE: / 'lv_foo is 2'.
  WHEN 3.
    WRITE: / 'lv_foo is 3'.
  WHEN OTHERS.
    WRITE: / 'lv_foo is something else'.
ENDCASE
```

PRÜFEN

`CHECK` ist eine einfache Anweisung, die einen logischen Ausdruck auswertet und den aktuellen Verarbeitungsblock beendet, wenn er falsch ist.

```
METHOD do_something.
  CHECK iv_input IS NOT INITIAL. "Exits method immediately if iv_input is initial

  "The rest of the method is only executed if iv_input is not initial
ENDMETHOD.
```

BEHAUPTEN

`ASSERT` wird in sensiblen Bereichen verwendet, in denen Sie absolut sicher sein möchten, dass eine Variable einen bestimmten Wert hat. Wenn sich die logische Bedingung nach `ASSERT` als falsch herausstellt, wird eine unhandliche Ausnahme (`ASSERTION_FAILED`) ausgelöst.

```
ASSERT 1 = 1. "No Problem - Program continues

ASSERT 1 = 2. "ERROR
```

`SWITCH` und `COND` bieten eine spezielle Form des bedingten Programmablaufs. Im Gegensatz zu `IF` und `CASE` repräsentieren sie unterschiedliche Werte, die auf einem Ausdruck basieren, anstatt Anweisungen auszuführen. Deshalb gelten sie als funktional.

COND

Wenn mehrere Bedingungen berücksichtigt werden müssen, kann `COND` die Arbeit erledigen. Die Syntax ist ziemlich einfach:

```
COND <type>(
    WHEN <condition> THEN <value>
    ...
    [ ELSE <default> | throw <exception> ]
).
```

Beispiele

```
" Set screen element active depending on radio button
screen-active = COND i(
    WHEN p_radio = abap_true THEN 1
    ELSE 0 " optional, because type 'i' defaults to zero
).

" Check how two operands are related to each other
" COND determines its type from rw_compare
rw_compare = COND #(
    WHEN op1 < op2 THEN 'LT'
    WHEN op1 = op2 THEN 'EQ'
    WHEN op1 > op2 THEN 'GT'
).
```

SCHALTER

`SWITCH` ist ein übersichtliches Werkzeug zum Abbilden von Werten, da es nur die Gleichheit prüft und daher in manchen Fällen kürzer als `COND` ist. Bei einer unerwarteten Eingabe kann auch eine Ausnahme ausgelöst werden. Die Syntax ist etwas anders:

```
SWITCH <type>(
    <variable>
    WHEN <value> THEN <new_value>
    ...
    [ ELSE <default> | throw <exception> ]
).
```

Beispiele

```
DATA(lw_language) = SWITCH string(
    sy-langu
    WHEN 'E' THEN 'English'
    WHEN 'D' THEN 'German'
    " ...
    ELSE THROW cx_sy_conversion_unknown_langu( )
).
```

Steuerungsflussanweisungen online lesen:
https://riptutorial.com/de/abap/topic/7289/steuerungsflussanweisungen

Kapitel 14: Unit-Tests

Examples

Struktur einer Testklasse

Testklassen werden als lokale Klassen in einem speziellen Unit-Test-Include erstellt.

Dies ist die Grundstruktur einer Testklasse:

```
CLASS lcl_test DEFINITION
               FOR TESTING
               DURATION SHORT
               RISK LEVEL HARMLESS.

PRIVATE SECTION.
  DATA:
    mo_cut TYPE REF TO zcl_dummy.

  METHODS:
    setup,

  "********* 30 chars *********|
  dummy_test                    for testing.
ENDCLASS.

CLASS lcl_test IMPLEMENTATION.
  METHOD setup.
    CREATE OBJECT mo_cut.
  ENDMETHOD.

  METHOD dummy_test.
    cl_aunit_assert=>fail( ).
  ENDMETHOD.
ENDCLASS.
```

Jede mit `FOR TESTING` deklarierte Methode ist ein `FOR TESTING` . `setup` ist eine spezielle Methode, die vor jedem Test ausgeführt wird.

Getrennter Datenzugriff von der Logik

Ein wichtiges Prinzip für das Testen von Einheiten besteht darin, den Datenzugriff von der Geschäftslogik zu trennen. Eine effiziente Technik hierfür ist das Definieren von Schnittstellen für den Datenzugriff. Ihre Hauptklasse verwendet immer einen Verweis auf diese Schnittstelle, anstatt direkt Daten zu lesen oder zu schreiben.

Im Produktionscode erhält die Hauptklasse ein Objekt, das den tatsächlichen Datenzugriff umschließt. Dies kann eine select-Anweisung, ein Aufruf von Funktionsmudulen, alles wirklich sein. Der wichtige Teil ist, dass diese Klasse nichts anderes ausführen sollte. Keine Logik.

Wenn Sie die Hauptklasse testen, geben Sie ihr stattdessen ein Objekt, das statischen,

gefälschten Daten dient.

Ein Beispiel für den Zugriff auf die `SCARR` Tabelle

Datenzugriffsschnittstelle `ZIF_DB_SCARR`:

```
INTERFACE zif_db_scarr
  PUBLIC.
    METHODS get_all
      RETURNING
        VALUE(rt_scarr) TYPE scarr_tab .
ENDINTERFACE.
```

Gefälschte Datenklasse und Testklasse:

```
CLASS lcl_db_scarr DEFINITION.
  PUBLIC SECTION.
    INTERFACES: zif_db_scarr.
ENDCLASS.

CLASS lcl_db_scarr IMPLEMENTATION.
  METHOD zif_db_scarr~get_all.
    " generate static data here
  ENDMETHOD.
ENDCLASS.

CLASS lcl_test DEFINITION
  FOR TESTING
  DURATION SHORT
  RISK LEVEL HARMLESS.

  PRIVATE SECTION.
    DATA:
      mo_cut TYPE REF TO zcl_main_class.

    METHODS:
      setup.
ENDCLASS.

CLASS lcl_test IMPLEMENTATION.
  METHOD setup.
    DATA: lo_db_scarr TYPE REF TO lcl_db_scarr.

    CREATE OBJECT lo_db_scarr.

    CREATE OBJECT mo_cut
      EXPORTING
        io_db_scarr = lo_db_scarr.
  ENDMETHOD.
ENDCLASS.
```

Die Idee hier ist, dass `ZCL_MAIN_CLASS` im Produktionscode ein `ZIF_DB_SCARR` Objekt `ZIF_DB_SCARR` , das ein `SELECT ZIF_DB_SCARR` und die gesamte Tabelle zurückgibt, während der `ZIF_DB_SCARR` gegen ein statisches Dataset ausgeführt wird, das genau dort im `ZCL_MAIN_CLASS ZIF_DB_SCARR` definiert ist.

Unit-Tests online lesen: https://riptutorial.com/de/abap/topic/3999/unit-tests

Kapitel 15: Vorlagenprogramme

Syntax

- CLASS DEFINITION ABSTRACT FINAL macht die Programmklasse im Wesentlichen statisch, da Instanzmethoden niemals verwendet werden könnten. Die Absicht ist, die Klasse minimal zu halten.

Examples

OO-Programm mit wesentlichen Ereignismethoden

```
REPORT z_template.

CLASS lcl_program DEFINITION ABSTRACT FINAL.

  PUBLIC SECTION.

    CLASS-METHODS start_of_selection.
    CLASS-METHODS initialization.
    CLASS-METHODS end_of_selection.

ENDCLASS.

CLASS lcl_program IMPLEMENTATION.

  METHOD initialization.

  ENDMETHOD.

  METHOD start_of_selection.

  ENDMETHOD.

  METHOD end_of_selection.

  ENDMETHOD.

ENDCLASS.

INITIALIZATION.

  lcl_program=>initialization( ).

START-OF-SELECTION.

  lcl_program=>start_of_selection( ).

END-OF-SELECTION.

  lcl_program=>end_of_selection( ).
```

Vorlagenprogramme online lesen: https://riptutorial.com/de/abap/topic/10552/vorlagenprogramme

Kapitel 16: Zeichenketten

Examples

Literale

ABAP bietet drei verschiedene Operatoren zur Deklaration von String- oder Char-like-Variablen an

Symbole	Interner Typ	Länge	Name
'...'	C	1-255 Zeichen	Textfeldliterale
"..."	CString	0-255 Zeichen	Textzeichenfolgenliterale
| ... |	CString	0-255 Zeichen	Vorlagenliterale

Beachten Sie, dass der Längenbereich nur für hart codierte Werte gilt. Intern haben `CString` Variablen eine beliebige Länge, während Variablen vom Typ `C` immer eine feste Länge haben.

String-Vorlagen

Stringvorlagen bieten eine bequeme Möglichkeit zum Mischen von literalen Strings mit Werten aus Variablen:

```
WRITE |Hello, { lv_name }, nice to meet you!|.
```

Es kann auch Dinge wie Datumsangaben formatieren. So verwenden Sie das Datumsformat des angemeldeten Benutzers:

```
WRITE |The order was completed on { lv_date DATE = USER } and can not be changed|.
```

Funktionale Methodenaufrufe und -ausdrücke werden unterstützt:

```
WRITE |Your token is { to_upper( lv_token ) }|.
WRITE |Version is: { cond #( when lv_date < sy-datum then 'out of date' else 'up to date' )
}|.
```

> **Beachtung!** Das direkte Implementieren temporärer Ergebnisse (wie Methodenaufrufe) in Stringvorlagen kann zu massiven Leistungsproblemen führen (lesen Sie hier mehr darüber). Die Verwendung in selten ausgeführten Anweisungen ist zwar in Ordnung, führt jedoch dazu, dass sich Ihr Programm in Schleifen schnell verlangsamt.

Zeichenketten verketten

`CONCATENATE`

und char-like-Variablen können mit `CONCATENATE` Befehl ABAP `CONCATENATE` verkettet werden. Eine zusätzliche Variable zum Speichern der Ergebnisse ist erforderlich.

Beispiel:

```
CONCATENATE var1 var2 var3 INTO result.
"result now contains the values of var1, var2 & var3 stringed together without spaces
```

Stenografie

Neuere Versionen von ABAP bieten eine sehr kurze Variante der Verkettung mit && (Verkettungsoperator).

```
DATA(lw_result) = `Sum: ` && lw_sum.
```

> **Beachtung!** Es lohnt sich zu bemerken, dass die Verwendung von temporären Ergebnisse in Kombination mit dem Chaining Betreiber innerhalb von Schleifen können aufgrund der wachsenden Kopie Anweisungen zu massiven Performance - Problemen führen (lesen Sie mehr darüber hier).

Zeichenketten online lesen: https://riptutorial.com/de/abap/topic/3531/zeichenketten

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit ABAP	Christian, Community, gkubed, Jagger, mkysoft
2	ABAP GRID List Viewer (ALV)	Achuth hadnoor, gkubed
3	ABAP-Objekte	Community, Michał Majer, Thomas Matecki
4	Bemerkungen	4444, Christian, gkubed
5	Datenerklärung	Christian, gkubed
6	Dynamische Programmierung	Community, gkubed
7	Interne Tabellen	Community, gkubed, Michał Majer, Rahul Kadukar, Thorsten Niehues
8	Nachrichtenklassen / MESSAGE-Schlüsselwort	gkubed
9	Öffnen Sie SQL	AKHIL RAJ, gkubed
10	Regeln der Namensgebung	mkysoft
11	Reguläre Ausdrücke	AKHIL RAJ, gkubed, maillard
12	Schleifen	Christian, Community, gkubed, Stu G
13	Steuerungsflussanweisungen	Community, gkubed, maillard
14	Unit-Tests	maillard
15	Vorlagenprogramme	nath
16	Zeichenketten	Achuth hadnoor, Community, maillard, nexxus, Suncatcher