**ABAP dialog programming**

[Содержание](#Содержание)

[Ресурсы](#Ресурсы)

**Ресурсы**

[Содержание](#Содержание)

# [Create GUI Status](#Create_GUI_Status)

# [LOOP - WITH CONTROL](#LOOP_WITH_CONTROL)

[Table Control](#Table_Control)

# **ABAP Blog Все разделы – Dynpro**

<https://abap-blog.ru/abap-notes/category/dynpro/>

# **Create ABAP dialog screen dynpro Table Control within SAP**

<https://www.trailsap.com/dev/abap/dialog/tabcontrol/?topic=tc_basic>

Структура ***/bic/abdcrrts02***

@EndUserText.label : *'Структура для системы отчетов RSDRI для хранилища данных BDC'*

@AbapCatalog.enhancementCategory : #NOT\_EXTENSIBLE

define type /bic/vbdcrrts02 {

key 0reqtsn : rspm\_request\_tsn not null;

key 0infoprov : rsinfoprov not null;

key 1rowcount : rssid not null;

0recordmode : rodmupdmod not null;

0date : abap.dats not null;

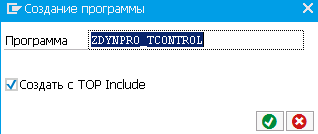
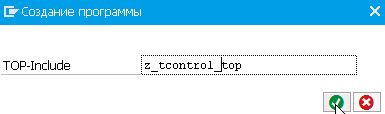
4bdcrrts0f\_n\_code : abap.numc(3) not null;

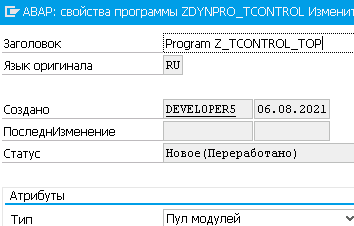
4bdcrrts0f\_c\_code : abap.char(3) not null;

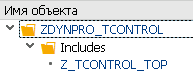
4bdcrrts0f\_dscr : abap.char(100) not null;

}

Программа ***zdynpro\_tcontrol***





*\*&---------------------------------------------------------------------\**

*\*& ПулМодул ZDYNPRO\_TABLE*

*\*&---------------------------------------------------------------------\**

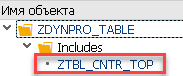
INCLUDE ZTBL\_CNTR\_TOP . *" global Data*

*\* INCLUDE ZTBL\_CNTR\_O01 . " PBO-Modules*

*\* INCLUDE ZTBL\_CNTR\_I01 . " PAI-Modules*

*\* INCLUDE ZTBL\_CNTR\_F01 . " FORM-Routines*

Модуль ***ZTBL\_CNTL\_TOP***



*\*&---------------------------------------------------------------------\**

*\*& Include ZTBL\_CNTR\_TOP ПулМодул ZDYNPRO\_TABLE*

*\*&---------------------------------------------------------------------\**

PROGRAM **ZTBL\_CNTR\_TOP**.

Tables: /bic/abdcrrts02.

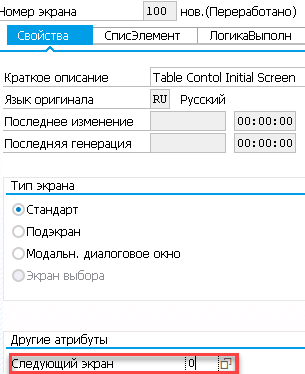
controls: tc100 type tableview using screen 100.

data: ok\_code type sy-ucomm.

data: it\_rates type standard table of /bic/vbdcrrts02 initial size 0,

wa\_rates type /bic/vbdcrrts02.

Экран 100



*\*&---------------------------------------------------------------------\**

*\*& Include ZTBL\_CNTR\_TOP ПулМодул ZDYNPRO\_TABLE*

*\*&*

*\*&---------------------------------------------------------------------\**

PROGRAM ZDYNPRO\_TABLE.

Tables: /bic/abdcrrts02.

controls: tc100 type tableview using screen 100.

data: ok\_code type sy-ucomm.

data: it\_rates type standard table of /bic/vbdcrrts02 initial size 0,

wa\_rates type /bic/vbdcrrts02.

The program you specify should be

1. an ***executable program*** (type 1),
2. a ***module pool*** (type **M**),
3. a ***function group*** (type **F**)

and must already exist.

**User Interface**

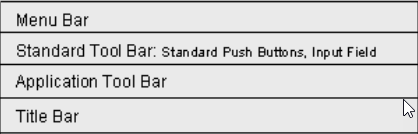
<https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-US/abenabap_dynpros_gui.htm>

Detailed inf. about dynpros can be found in the documentation *Classic Dynpro Programming* in SAP Help Portal.

*Note*

The classic dynpros are considered to be obsolete by SAP for application programs - for new developments, only [SAPUI5](javascript:call_link('abensapui5_glosry.htm')) or [Web Dynpro](javascript:call_link('abenweb_dynpro_glosry.htm')) should be used.

Each standard window contains a ***menu bar***, a ***standard toolbar***, and an ***application toolbar***.

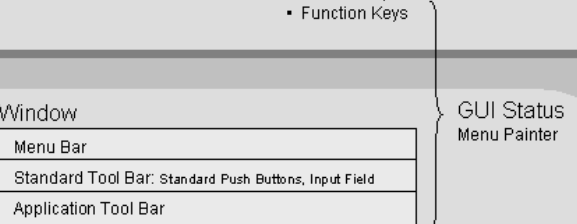


*Rem*

GUI windows that are displayed as a *modal dialog box* contain only an *application toolbar*.

The ***bars*** are *grouped together in a* *GUI status*.

The ***function keys*** are part of a GUI status.

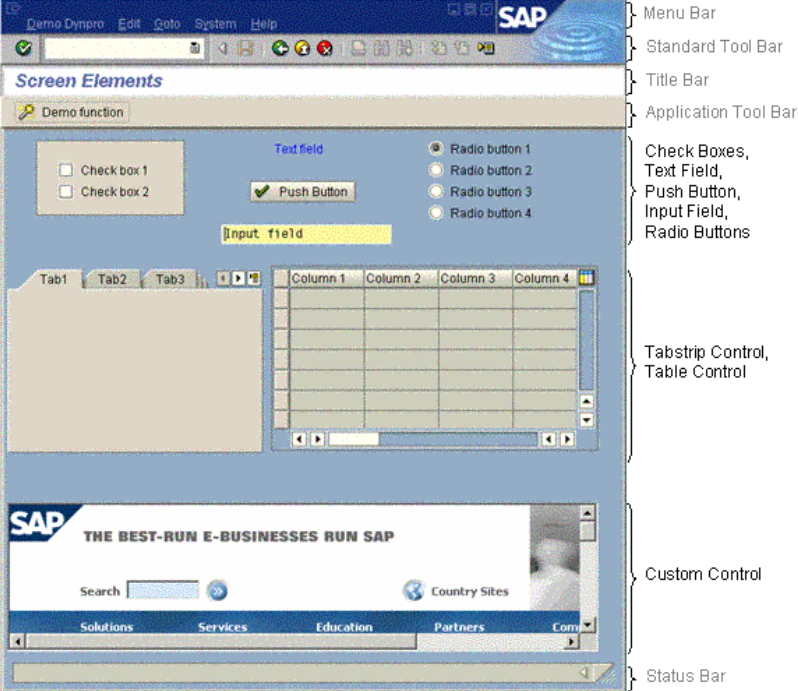


For the most part, ***the control elements of the user interface*** *are linked with the function codes*, which can be evaluated in the ABAP program - the *description of the current GUI status* can be found in the system field ***sy-pfkey***.

A GUI window in SAP GUI is complete when it has a ***title bar*** and a ***status bar***.

The **status bar** displays inf. that can be transmitted, among other things, during the execution of an ABAP program using the [***MESSAGE***](javascript:call_link('abapmessage.htm'))statement. In addition, it contains ***system inf****.* that can be displayed/hidden using an *icon on the right of the bar*.

**Screen and Screen Elements**



**Dynpro Fields**

***The data types of dynpro fields*** are determined either by reference

* to [***built-in data types***](javascript:call_link('abenddic_builtin_types.htm'))from ABAP Dictionary /except ***CLNT*** and ***FLTP/*** or
* to ***global fields***of the ABAP program.

All elementary data types *except* [*enumerated types*](javascript:call_link('abenenumerated_type_glosry.htm'))*are possible*.

At **PBO** time, all dynpro fields are transported at the end of PBO processing, with the exception of fields that are defined in table controls or in [***step loops***](javascript:call_link('abenstep_loop_glosry.htm')). The latter are processed in loops in the flow logic and *transported from the ABAP program to the dynpro after each loop execution*.

At **PAI** time,

* ***the content of all dynpro fields*** that do not belong to any [table control](javascript:call_link('abentable_control_glosry.htm')) or step loop and are not specified in any [FIELD](javascript:call_link('dynpfield.htm')) statement are transported *into the fields with the same name in the ABAP program*.
* The contents of the fields of ***a table control*** or ***step loop*** are transported *to the ABAP program* row by row or group by group *at the beginning of the respective loop run*.
* The fields that are specified in the ***FIELD*** statements of the dynpro flow logic *are transported upon execution of the corresponding FIELD statement*.
* Pure display elements such as ***text fields*** or ***frames*** are not associated with dynpro fields and *do not necessarily need a unique name*.
* When you create ***input and output fields by using fields from ABAP Dictionary***, the *system* usually also *creates field labels using texts from the dictionary*.
* A dynpro also knows the [***system fields***](javascript:call_link('abensystem_field_glosry.htm')) of the ABAP program. In contrast to the program, these system fields are exclusively addressed as **syst-name**.

***Screen numbers***

* All screen numbers above 9000 are reserved *for SAP's customers*.
* The number 1000 is reserved *for table screens and report selection screens*. *Initial screens of transactions* are often given a number whose last three digits are 100 /for example, 3100/.

**Screen fields**

When you create a screen, SAP automathic creates a ***field with the type OK***. The *function code* of the psuhbutton will be placed here when you push the button. However you have to supply the name of the OK field / for example OK\_CODE/.

You must also create a ***global variable*** in the program, to store the value of the OK field -

DATA:

*\* Global variable to store OK code*

ok\_code(4),

*\* Temporary store the value of the OK code*

save\_ok\_code(4).

**Deactivate a field on the screen**

MODULE status\_0001 OUTPUT.

SET PF-STATUS '0001'.

SET TITLEBAR '001'.

*\* Example of how deactivate a field on the screen*

*\* ZCOSTAFSTM-ZAFSTEMNR is the name of the screen field we want to deactivate.*

LOOP AT SCREEN.

CHECK screen-name = 'ZCOSTAFSTM-ZAFSTEMNR'.

screen-input = '0'.

MODIFY SCREEN.

ENDLOOP.

ENDIF.

ENDMODULE.

**Обработка команды пользователя**

**MODULE** *user\_command\_0001* INPUT.

*\* Here you can catch when user pushes a pushbutton*

*\* Save the OK code in save\_ok\_code and clear it*

save\_ok\_code = ok\_code.

CLEAR ok\_code.

CASE save\_ok\_code.

WHEN '0010'.

*LEAVE TO LIST-PROCESSING* AND *RETURN TO SCREEN* 0.

PERFORM my\_list.

WHEN '0020'. *CALL TRANSACTION* 'ZCO1'.

WHEN 'RETU'. *LEAVE TO SCREEN* '0000'.

ENDCASE.

ENDMODULE.

**AT USER-COMMAND**.

*\* Here you can catch when the user psuh a button on the menu bar or presses a function key*

CASE sy-ucomm.

WHEN 'OPRT'. *PERFORM* something.

ENDCASE.

# **Update values in dynpro fields**

Data: dyfields LIKE dynpread OCCURS 1 WITH HEADER LINE.

dyfields-fieldname = 'PDPLA-VON\_DATE'.

dyfields-fieldvalue = l\_datestr.

APPEND dyfields.

CALL FUNCTION 'DYNP\_VALUES\_UPDATE'

EXPORTING

dyname = sy-cprog

dynumb = sy-dynnr

TABLES

dynpfields = dyfields

EXCEPTIONS

invalid\_abapworkarea = 1

invalid\_dynprofield = 2

invalid\_dynproname = 3

invalid\_dynpronummer = 4

invalid\_request = 5

no\_fielddescription = 6

undefind\_error = 7

OTHERS = 8.

IF sy-subrc <> 0.

ENDIF.

### **Module pool programming - Example input,output field element**

### <http://saptechnicals.blogspot.com/2012/12/module-pool-programming-example.html>

ZUSERDIALOG\_01

### **Module pool programming -** **Create GUI Status example**

<http://saptechnicals.blogspot.com/2012/12/module-pool-programming-create-gui.html>

[Содержание](#Содержание)

# **LOOP - WITH CONTROL**

[Содержание](#Содержание)

***Syntax***

LOOP [AT itab [INTO wa] [CURSOR top\_line] [FROM n1] [TO n2]] WITH CONTROL contrl.

  ...

ENDLOOP.

***Effect***

The loop sequentially processes the visible rows of the table control /TC/ by executing one loop pass for reach table control row. *If the TC does not exist, the loop is ignored.* The statement block between LOOP and ENDLOOP can contain the keywords [FIELD](javascript:call_link('dynpfield.htm')), [MODULE](javascript:call_link('dynpmodule.htm')), [CHAIN](javascript:call_link('dynpchain.htm')), and [ENDCHAIN](javascript:call_link('dynpendchain.htm')) from the flow logic. *Nesting of loops is not possible*. Loops can be executed either with or without reference to an internal table.

If TC are defined in a dynpro, *one loop must be defined for each TC both in the* [*PBO*](javascript:call_link('abenpbo_glosry.htm')) *processing block and in the* [*PAI*](javascript:call_link('abenpai_glosry.htm')) *processing block*.

***System Fields***

Within the loop pass, the system field **sy-stepl** contains *the number of the current TC row*, starting at the top visible row. The system field **sy-loopc** contains *the total number of TC rows* displayed on the screen.

***Variant 1***

LOOP WITH CONTROL contrl.

  ...

ENDLOOP.

***Effect***

*If the addition* ***AT itab*** *is not specified*, the contents of the dynpro fields of the current row of the TC are transported in a loop pass from /at event PBO/ or to /at event PAI/ the data objects with the same names in the ABAP program.

In **PBO** processing, the *transport is performed at the end of the loop pass*; in **PAI** processing, *at the start of the loop pass*. The addition WITH CONTROL must be specified both at PBO and PAI.

***Example***

If a table control /TC/ *FLIGHT\_TAB* is defined on the screen of a dynpro, the associated dynpro flow logic may look like this. The loop is executed with reference to the internal table *spfli\_tab*.

***At PBO,*** the loop calls a dialog module *prepare\_tab* to fill the internal table. In the loop, no dialog module is called at PBO, because the TC in this case is filled automatically.

***At PAI***, a dialog module *modify\_tab* is called in the loop to save the changes the user entered in the TC to the internal table.

PROCESS BEFORE OUTPUT.   
   MODULE *prepare\_tab*.   
   *LOOP AT* *spfli\_tab* INTO spfli WITH CONTROL flight\_tab.   
   ENDLOOP.

PROCESS AFTER INPUT.

*LOOP AT spfli\_tab.*

    MODULE *modify\_tab.*

  ENDLOOP.

*Notes*

* *For dynpro fields of the TC* defined with a reference to flat structures in ABAP Dictionary, *the data objects with the same names in the ABAP program must be declared* in the same way as regular dynpro fields *using* [**TABLES**](javascript:call_link('abaptables.htm')); otherwise no data transport takes place.
* In the loop, dialog modules can be called to process the relevant data objects of the ABAP program. For example, *data from an internal table can be read at PBO and written back to the table at PAI*, after being processed on the screen.

***Variant 2***

LOOP AT itab CURSOR cur [INTO wa] [CURSOR top\_line] [FROM n1] [TO n2] WITH CONTROL contrl.

  ...

ENDLOOP.

***Effect***

*If the addition* ***AT itab*** *is specified*, the internal table itab of the associated ABAP program is processed sequentially in parallel to the loop processing of the TC. For each row of the TC, one row of the internal table is processed. *The internal table itab must be an index table.* The additions INTO, CURSOR, FROM, TO, and WITH CONTROL can be specified only at PBO and not at PAI. At PAI, the internal table is used to create a reference to the TC.

The addition INTO is used to specify a work area wa to which the current row of the internal table is assigned **at PBO** at the end of each loop pass*. If the addition* ***wa*** *is not specified, an internal table with header line must be used*, which is then used implicitly instead of wa. After the assignment, *the content of wa or of the header line is transported to fields with the same names in the current row of the TC*. The work area wa must be a global data object of the ABAP program that matches the row type of the internal table.

**At** the event **PAI,** only the work area wa or the header line of the internal table is filled with the content of the TC rows at the beginning of each loop pass. *The content of the internal table is not modified automatically.*

The additions CURSOR, FROM, and TO are possible but not necessary, because the **TC** are designed to be *controlled by the structure of type CXTAB\_CONTROL created using* [**CONTROLS**](javascript:call_link('abapcontrols_tableview.htm')) *in the ABAP program*.

Here, ***top\_line*** of component TOP\_LINE corresponds to this structure while *the number of rows to be displayed* can be controlled using the component ***LINES*** instead of n1 and n2. *If n1 is still specified for table controls*, the content of component ***CURRENT\_LINE*** *is calculated as* follows, differing from the method shown at [CONTROLS](javascript:call_link('abapcontrols_tableview.htm')) - sy-stepl+ (TOP\_LINE - 1) + (n1 - 1).

*Notes*

Between LOOP and ENDLOOP, no dialog module must be called at PBO to read the data from the internal table. At PAI, however, this is necessary provided the transported data needs to be evaluated. For example, the internal table can be modified in accordance with the user entries.

***Example*** *-* Table Control with Modifications

PROCESS BEFORE OUTPUT.

MODULE STATUS\_0100.

loop at itab INTO DEMO\_CONN with control flights.

endloop.

PROCESS AFTER INPUT.

MODULE CANCEL AT EXIT-COMMAND.

loop at itab.

module read\_table\_control.

endloop.

MODULE USER\_COMMAND\_0100.

REPORT **demo\_dynpro\_tabcont\_loop\_at**.

CONTROLS *flights* TYPE TABLEVIEW USING SCREEN 100.

DATA: cols LIKE LINE OF flights-cols,  
       lines TYPE i.  
 DATA: ok\_code TYPE sy-ucomm,  
       save\_ok TYPE sy-ucomm.  
 DATA: itab TYPE TABLE OF demo\_conn.  
 TABLES demo\_conn.

*“ spfli -> @itab* SELECT \* FROM spfli INTO CORRESPONDING FIELDS OF TABLE *@itab*      

*“ flights-cols -> screen-input = '0'*

LOOP AT flights-cols INTO cols WHERE index GT 2.  
   cols-screen-input = '0'.  
   MODIFY flights-cols FROM cols INDEX sy-tabix.  
 ENDLOOP.  
  
 CALL SCREEN 100.  
  
 MODULE ***status\_0100*** OUTPUT.  
   SET PF-STATUS 'SCREEN\_100'.  
   DESCRIBE TABLE itab LINES lines.  
   flights-lines = lines.  
 ENDMODULE.

MODULE ***cancel*** INPUT.  
   LEAVE PROGRAM.  
 ENDMODULE.

MODULE ***read\_table\_control*** INPUT.

  MODIFY itab FROM demo\_conn INDEX flights-*current\_line*.

ENDMODULE.

MODULE ***user\_command\_0100*** INPUT.

  save\_ok = ok\_code.

  CLEAR ok\_code.

  CASE save\_ok.

    WHEN 'TOGGLE'.

*“ The first two columns are lead columns.*

LOOP AT flights-cols INTO cols WHERE index GT 2.

        IF  cols-screen-input = '0'.  cols-screen-input = '1'.

        ELSEIF  cols-screen-input = '1'. cols-screen-input = '0'.

        ENDIF.

        MODIFY flights-cols FROM cols INDEX sy-tabix.

      ENDLOOP.

    WHEN 'SORT\_UP'.

*“ The selection column is assigned to the component MARK, of the type CHAR, length 1, from the*

*“ structure DEMO\_CONN.*

      READ TABLE flights-cols INTO cols WITH KEY *selected = 'X'*.

      IF sy-subrc = 0.

*“ The name of the sort criterion in the SORT statement is determined dynamically from the component*

*“ cols-screen-name. The prefix /demo\_conn-/ must be removed by specifying an offset. After the sort,*

*“ the selection is canceled and the component selected in the table flights-cols is assigned a blank*

*“ character.*

SORT itab STABLE BY (cols-screen-name+10) ASCENDING.

        cols-selected = ' '.

        MODIFY flights-cols FROM cols INDEX sy-tabix.

      ENDIF.

    WHEN 'SORT\_DOWN'.

      READ TABLE flights-cols INTO cols WITH KEY selected = 'X'.

      IF sy-subrc = 0.

        SORT itab STABLE BY (cols-screen-name+10) DESCENDING.

        cols-selected = ' '.

        MODIFY flights-cols FROM cols INDEX sy-tabix.

      ENDIF.

    WHEN 'DELETE'.

*“ First the system checks in flights-cols whether the fields of the table control are ready for input.*

*“ Then all selected rows are deleted in a loop using the internal table itab. Since the table control is*

*“ filled completely from the internal table itab during the PBO loop, the rows are also deleted on the*

*“ screen.*

      READ TABLE flights-cols INTO cols WITH KEY *screen-input = '1'*.

      IF sy-subrc = 0.

        LOOP AT itab INTO demo\_conn WHERE mark = 'X'.

          DELETE itab.

        ENDLOOP.

      ENDIF.

  ENDCASE.

ENDMODULE.

# **Table Control**

[Содержание](#Содержание)

**The Structure CXTAB\_COLUMN**

**SCREEN** - *structure for the attributes* of the screen element *of the current column*. The components can be set for the values described there either directly or using MODIFY SCREEN. ***MODIFY SCREEN*** *overwrites a direct assignment*.

Current position of the column in the TC. The start value is taken from the definition of the TC in the dynpro. Is set to current value at time of PAI.

**SELECTED** /'X' or ' '/ - whether or not column is selected. Is set to current value at time of PAI.

**VISLENGTH** - visible length of the column. Start value is taken from the definition of the table control in dynpro.

**INVISIBLE** /'X' or ' '/ - whether or not the column is visible in the TC.

# **Table Control Example ABAP**

<https://sapintegrationhub.com/abap/ddic/table-control-example-abap/>

# **Create ABAP dialog screen dynpro Table Control within SAP**

<https://www.trailsap.com/dev/abap/dialog/tabcontrol/?topic=tc_basic>

In BSP we must work with ABAP OO, which means that you cannot declare an internal table with a header. You need the internal table and a work area.

Declare a work area like this

data wa\_kna1 type kna1.

and then read data like this

select \* from kna1 into wa\_kna1 where kunnr in it2\_range

*“some code here*

endselect.

or you can do it like this

select \* from kna1 into corresponding fields of *table* it\_kna1

where kunnr in it2\_range

### **Add extra tabs to the standard transaction for sales and access them through menu bar**

<http://sapabapcentral.blogspot.com/2017/12/add-extra-tabs-to-standard-transaction.html>

### [**ABAP - добавление своей кнопки в TOOLBAR ALV GRID**](http://n-memos.blogspot.com/2011/04/abap-toolbar-alv-grid.html)

<http://n-memos.blogspot.com/2011/04/abap-toolbar-alv-grid.html>

[Nokia 5.4 4GB/64GB Dual Sim / TA-1337 (синий)](https://www.21vek.by/mobile/544gb64gbdualsimta1337_nokia.html)

Или

[Samsung Galaxy A12 64GB / SM-A125FZKVSER (черный)](https://www.21vek.by/mobile/galaxya1264gbsma125fzkvser_samsung.html)

Nokia 5.4 или Samsung Galaxy A12

ZDYNPRO\_TCONTROL\_03