BC SAPscript: Printing with Forms

HELP.BCSRVSCRPROG

Release 4.6B

Copyright

Icons

Icon	Meaning
⚠	Caution
💬	Example
➡	Note
🧭	Recommendation
SYN	Syntax
💡	Tip

Contents

BC SAPscript: Printing with Forms

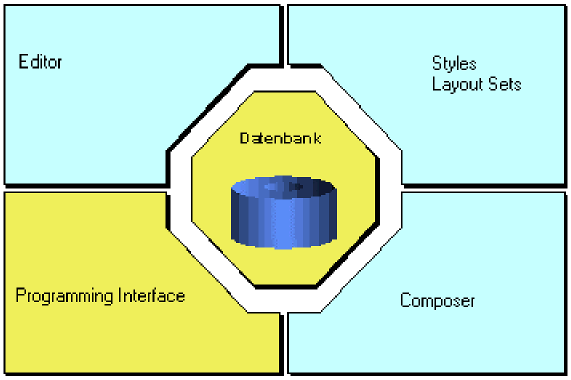
BC - SAPscript: Printing with Forms

Overview

To use forms efficiently, it is essential that you understand the interdependencies between the individual components of SAPscript.

SAPscript comprises these five components:

- an editor for entering and editing the lines of a text. The application transactions automatically call this editor if the user decides to maintain texts that concern an application object.
- styles and forms for print layout. These are created independent of the individual texts using the corresponding maintenance transactions and are allocated to the texts later.
- the composer as central output module. Its task is to prepare a text for a certain output device by including the corresponding formatting information. This information comes from the style and form allocated to the text. The SAPscript composer is invisible on the outside.
- a programming interface that allows you to include SAPscript components into your own application programs and to control the output of forms from within the programs.
- several database tables for storing texts, styles and forms.



In short, form maintenance means to allocate to a text document a form that contains the information on how to layout the text (formats, fonts, layout, and so on). The print program retrieves the required data from the form and from the database and controls the output. You use certain function modules to activate the SAPscript composer, which is responsible for processing the form.

Not every user of the SAP system works with every component of SAPscript. Depending on the task, a user is confronted with different components.

- Accounting clerks create texts concerning materials, orders, customers, vendors, and so on. They usually know only the SAPscript editor.
- Another user may be responsible for the print layout and will use the transactions for maintaining styles and forms.
- A developer who integrates SAPscript into his own applications or who wants to create print output using forms, sees SAPscript from the programming interface view point.

Overview

This documentation explains the basic principles of printing texts using forms. It describes in detail the interaction of form and print program and offers examples for better understanding.

Building on this basic knowledge, the documentation then describes the programming interface which allows you to integrate the word processing functionality offered by SAPscript into ABAP programs. This interface is a collection of ABAP function modules, different data structures, and control tables.

Printing texts using forms [Page 12]

The programming interface [Page 58]

Printing Texts Using Forms

This documentation describes in detail the basic principles of printing texts using forms and the interactions of the different SAPscript components.

Structure of a Form

To output documents using the programming interface [Page 58], R/3 application programs need so-called **forms.** In SAPscript a form describes the layout of the individual print pages and uses text elements [Page 16] to supply definable output blocks, which a print program [Page 31] can call. General application forms are orders, order acknowledgments, invoices, urging letters, and so on (Example [Page 15]).

To create, display, and change SAPscript forms, you use a special maintenance transaction (Form: Request). To call this transaction from the initial R/3 screen, choose *Tools* → *Word processing* → *Form* (or call transaction SE71 directly). For more detailed information on maintaining forms, see the documentation *Style and Form Maintenance*.

Usually a SAPscript form consists of the following objects:

Header Data [Page 14]

Paragraph and Character Formats [Page 17]

Windows and Text Elements [Page 18]

Pages [Page 19]

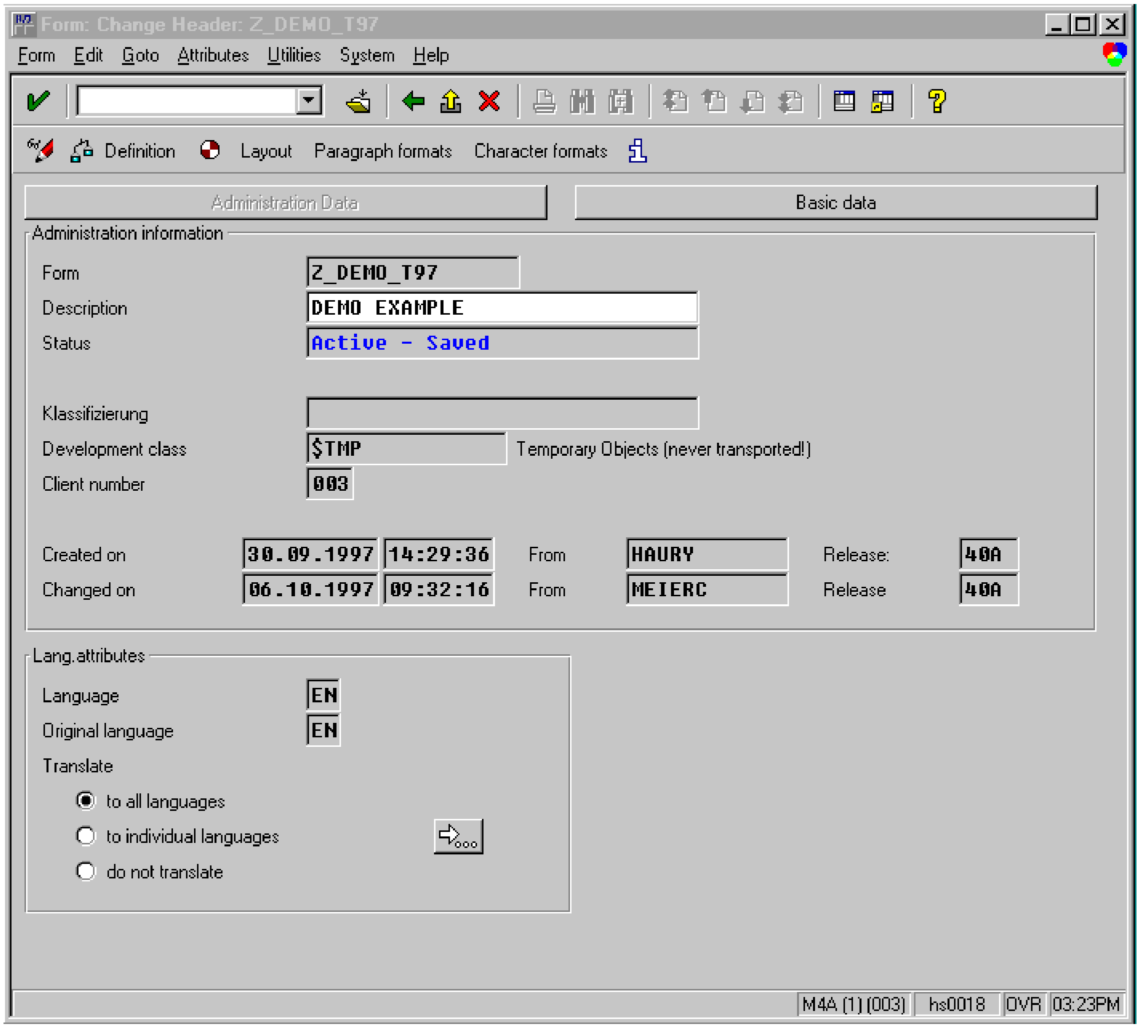
Page Windows [Page 24]

Form documentation [Ext.]

Header Data

The header data of a form consists of global data, such as the page format used, the page orientation, or the initially used font.You maintain these data in the *Basic settings* of the header data. The header data also include the name of the form, its description, the form class, and the status. To see them, branch to the administration data (pushbutton).

If you display or edit the form header in the form maintenance transaction, the screen looks more or less like the one below. Depending on whether you use the graphical (32bit platform Windows 95 or Windows NT 4.0 required!) or the traditional alphanumeric Form Painter, you can design the layout with the mouse (pushbutton *Layout*) or by defining *Windows, Pages* and *Page windows*. Below, you see the interface of the graphical Form Painter.



Administration information

Form	Z_DEMO_T97				
Description	DEMO EXAMPLE				
Status	Active - Saved				
Klassifizierung					
Development class	$TMP Temporary Objects (never transported!)				
Client number	003				
Created on	30.09.1997 14:29:36	From	HAURY	Release:	40A
Changed on	06.10.1997 09:32:16	From	MEIERC	Release	40A

Lang.attributes

Language EN

Original language EN

Translate

● to all languages

○ to individual languages

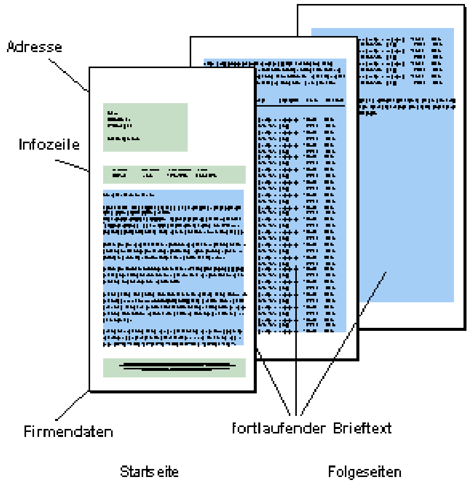
○ do not translate

M4A (1) (003) hs0018 OVR 03:23PM

Layout Example

The figure below shows a simple example of an invoice form created using SAPscript. Each form consists of a **start page** and any number of **subsequent pages**, depending on the length of the letter text. In this example, the start page consists of an area for outputting the address, an information window containing reference data and the date, a window [Page 21] containing company-related data, and a main window for the actual letter text. This window for the letter text appears on the subsequent pages as well, and, in addition, a window for page numbering.

The window for the text body differs from the other windows. Whenever this window on one page [Page 22] is full, the remaining text is automatically output on the subsequent page. The window thus controls the page break. There can be only one window that triggers a page break. Such a window in SAPscript is called **main window**.



Example of an invoice form in SAPscript

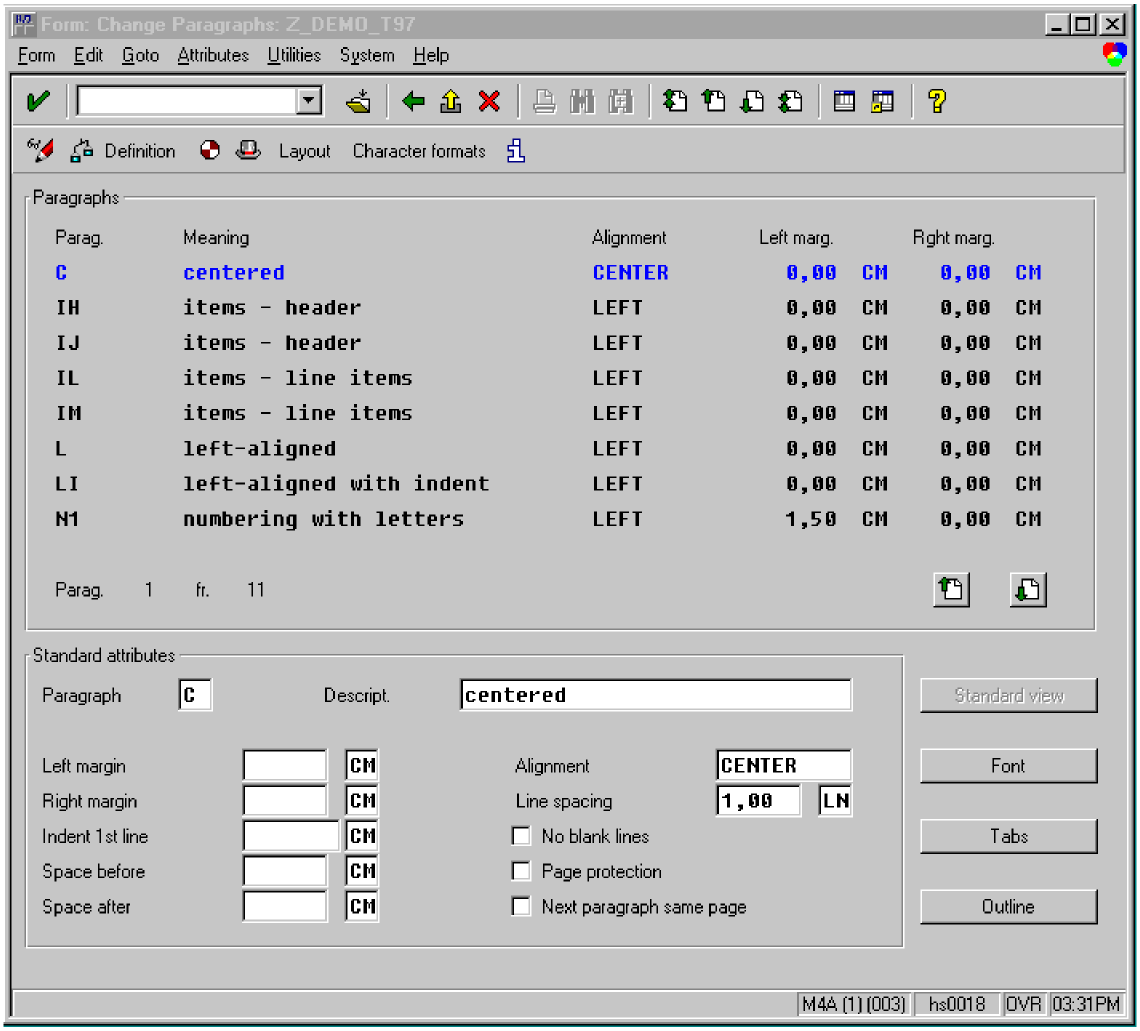
Text Element

Text elements in SAPscript are the individual text components of a form. In the different windows, you can define text elements with different attributes. For printout, the print program accesses them. Text elements can also contain variables (symbols) and SAPscript control statements.

See also: Text Elements of a Form [Page 26]

Paragraph and Character Formats

To format texts in forms, you need paragraph and character formats. You define them in the form itself. If you are working with the line editor, the paragraphs you define here appear in the possible values list (`F4`) for the format column beside the system-defined standard paragraphs. If you are using the WYSIWYG PC editor (32bit platforms Windows 95 or Windows NT 4.0 required), the formats defined appear as pushbuttons. The figure below shows an example of how to maintain a paragraph within transaction SE71 (Form: Request). Use the pushbuttons *Font, Tabs,* and *Outline* to refine paragraph definitions according to your requirements.


Form: Change Paragraphs: Z_DEMO_T97
Form Edit Goto Attributes Utilities System Help
Definition
Layout
Character formats
Paragraphs
Standard attributes
Standard view
Font
Tabs
Outline
M4A (1) (003) hs0018 OVR 03:31PM

Parag.	Meaning	Alignment	Left marg.		Right marg.	
C	centered	CENTER	0,00	CM	0,00	CM
IH	items - header	LEFT	0,00	CM	0,00	CM
IJ	items - header	LEFT	0,00	CM	0,00	CM
IL	items - line items	LEFT	0,00	CM	0,00	CM
IM	items - line items	LEFT	0,00	CM	0,00	CM
L	left-aligned	LEFT	0,00	CM	0,00	CM
LI	left-aligned with indent	LEFT	0,00	CM	0,00	CM
N1	numbering with letters	LEFT	1,50	CM	0,00	CM

Parag. 1 fr. 11

Paragraph: C   Descript.: centered

Left margin: CM   Alignment: CENTER
Right margin: CM   Line spacing: 1,00 LN
Indent 1st line: CM   □ No blank lines
Space before: CM   □ Page protection
Space after: CM   □ Next paragraph same page

For more information on how to create paragraph formats, see the R/3 online help documentation BC - SAPscript - Style and Form Maintenance [Ext.].

Windows and Text Elements

Forms usually consist of individual text areas (address, date, footer, and so on). To provide these areas with texts, you must define the areas first as output areas. Then you can print the appropriate texts in these output areas, controlled by the print program. SAPscript calls such an output area a **window**. To refer to windows via the programming interface, each window must have a unique name.

Frequently used window names in application forms are ADDRESS, SENDER, MAIN, or PAGE.

You can assign texts to each form window. These so-called text elements [Page 16] are part of the form and stored together with the other form elements. Text elements also receive names. You use these names to refer to the respective text elements via the programming interface.

The control commands and variable symbols used in the editor when creating text elements correspond to the SAPscript notation used to maintain long texts.

You can create each **window** only once on each page [Page 22], except the main window, which may appear up to 99 times on each page (for example, for printing labels). Windows may overlap, which can be of importance in certain output situations.

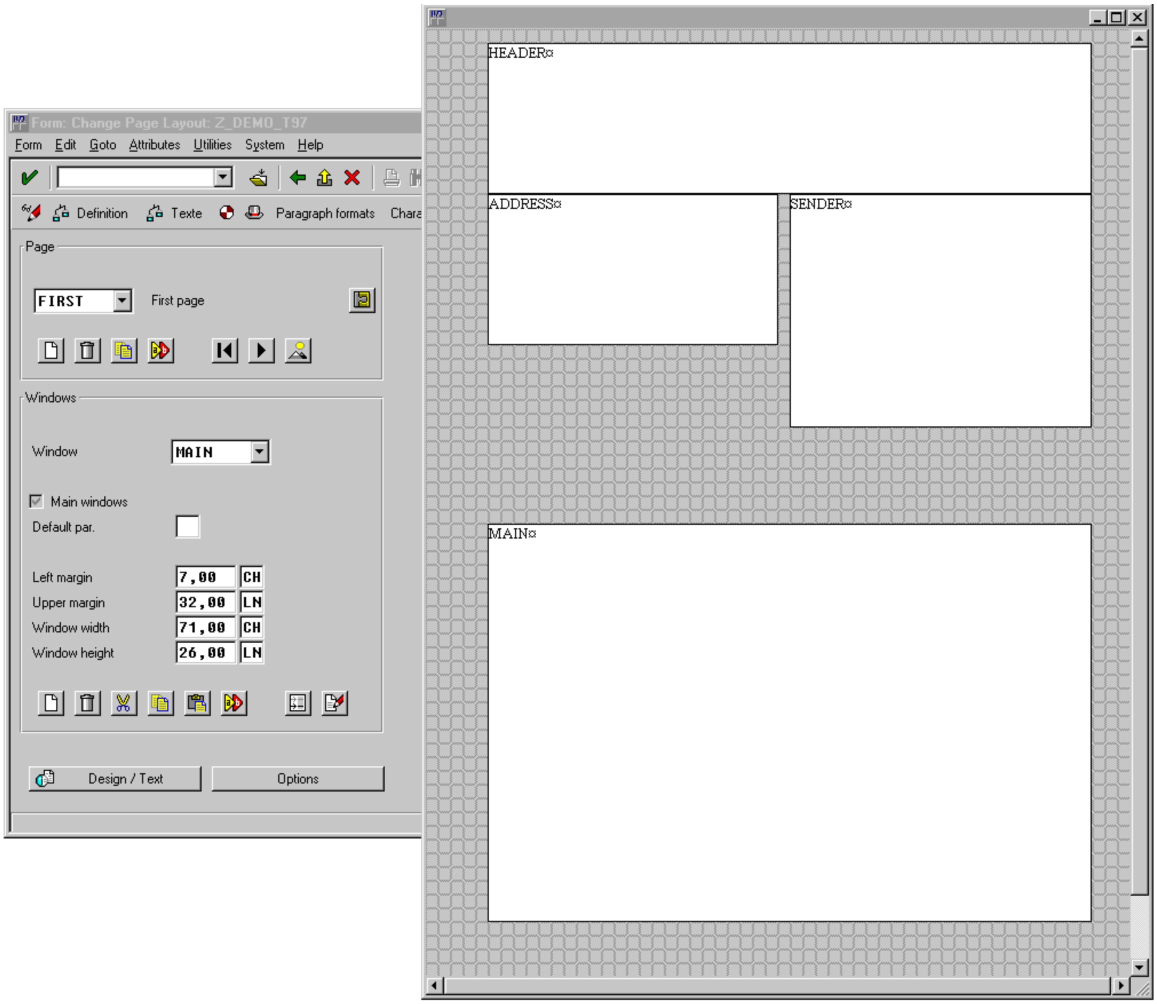
For more information on editing SAPscript texts, see the R/3 online help documentation *Word Processing in the SAP*script *Editor*.

Pages

The individual **pages** of a document often have different layouts: The first page of an invoice differs from the subsequent pages, on which you need not repeat general information, such as address or customer data. Just as the text elements of a window, the pages also have names. You may need these names to specify the subsequent page in case of a page break. The more variable you want the layout of a document to be, the more different pages you will define in the SAPscript form.

The figure below shows an example of how to maintain a page within transaction SE71 (Form: Request). You see the graphical Form Painter, which you can use under the frontend operating systems Windows 95 and Windoes NT 4.0.

For more information on the graphical Form Painter, see the SAP online documentation *BC - Style and Form Maintenance*.



Form: Change Page Layout: Z_DEMO_T97

Page: FIRST First page

Window: MAIN

☑ Main windows

Default par. □

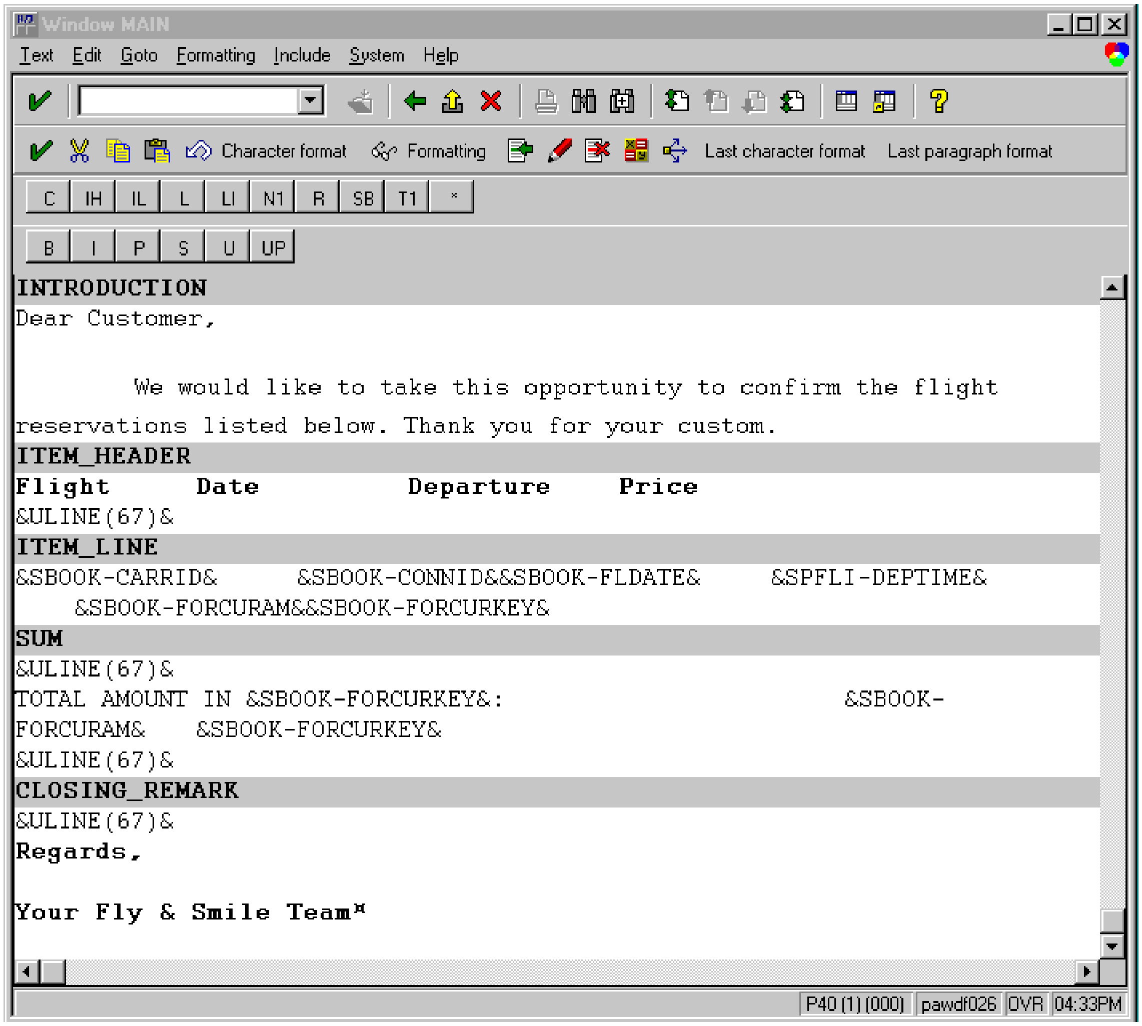
Left margin 7,00 CH

Upper margin 32,00 LN

Window width 71,00 CH

Window height 26,00 LN

Text Elements in the PC Editor



```
INTRODUCTION
Dear Customer,

        We would like to take this opportunity to confirm the flight
reservations listed below. Thank you for your custom.
ITEM_HEADER
Flight      Date           Departure     Price
&ULINE(67)&
ITEM_LINE
&SBOOK-CARRID&     &SBOOK-CONNID&&SBOOK-FLDATE&     &SPFLI-DEPTIME&
    &SBOOK-FORCURAM&&SBOOK-FORCURKEY&
SUM
&ULINE(67)&
TOTAL AMOUNT IN &SBOOK-FORCURKEY&:                         &SBOOK-
FORCURAM&   &SBOOK-FORCURKEY&
&ULINE(67)&
CLOSING_REMARK
&ULINE(67)&
Regards,

Your Fly & Smile Team¤
```

Window

Documents usually consist of different areas containing different texts (date, address, and so on). SAPscript calls such an output area **window**. These windows can later be positioned on different pages. Filling the different windows with the corresponding texts, is controlled by the print program or the composer, respectively.

Pages

Different pages of a document may have different layouts. For example, the first page of an invoice contains the customer address and letter text. The subsequent pages contain the actual invoice data, the order number, and so on. To be able to call the correct subsequent page after a page break, each individual page in SAPscript must have a name.

Page Window

A page window is the definition of a rectangular output area on the output medium (for example, DIN A4 paper page), determined by the left upper edge and the hight and width of the area.

By defining page windows, you determine which windows you want to appear on a page, their sizes and their positions.

➡

Starting with Release 4.0, the definition of page windows is required for the alphanumeric Form Painter only. If you use the graphical Form Painter, you can use the mouse to place the windows on the page (drag & drop, cut & paste). For more information on the graphical Form Painter, refer to *BC - Style and Form Maintenance*.

Page Windows

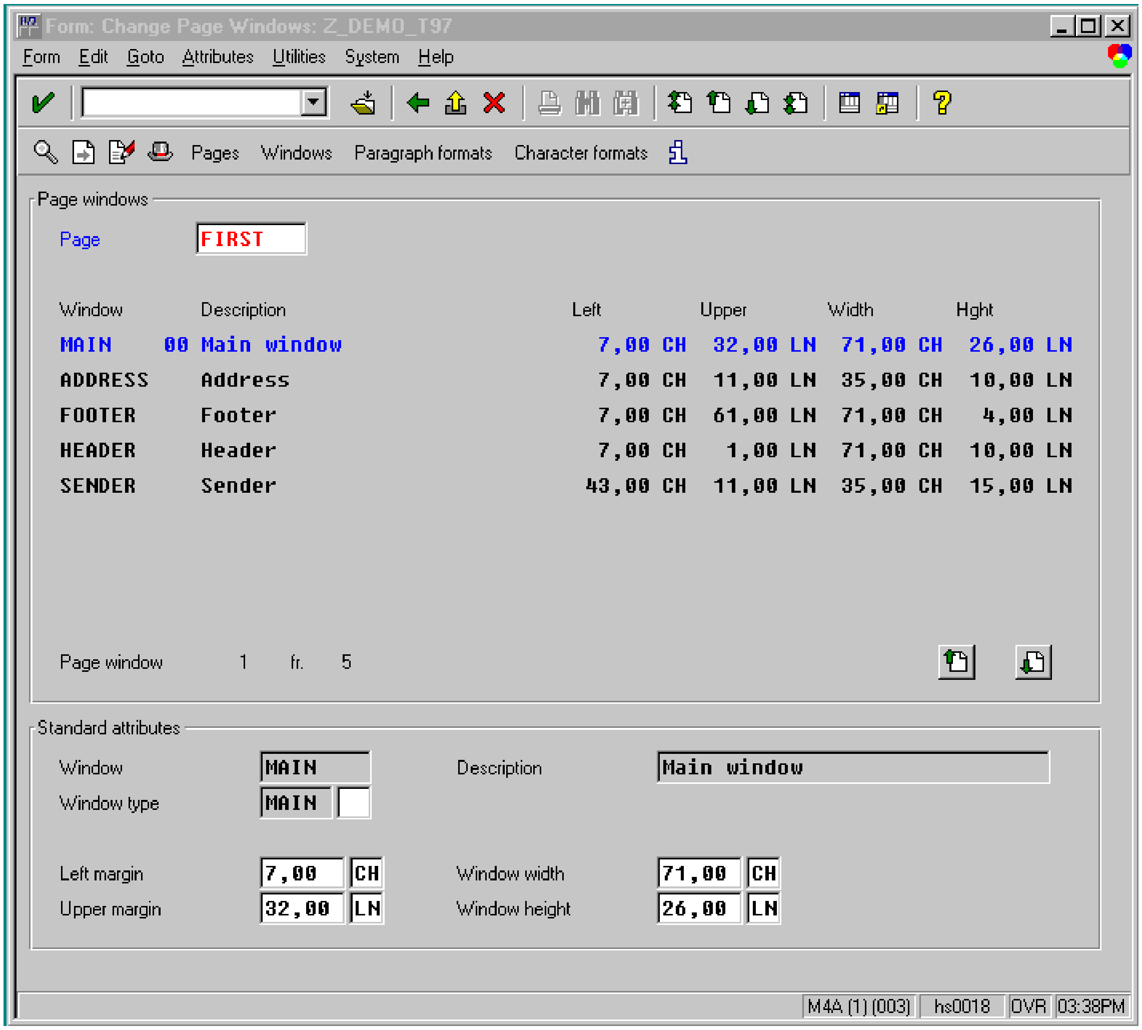
Starting with Release 4.0, the definition of page windows is required for the alphanumeric Form Painter only. If you use the graphical Form Painter, you can use the mouse to place the windows on the page (drag & drop, cut & paste). For more information on the graphical Form Painter, refer to *BC - Style and Form Maintenance*.

If you use the alphanumeric Form Painter, proceed as follows:

When defining windows and pages, you do not yet determine the position and spacing of the texts to be output. To do this, you combine a window and a form page to create a so-called **page window**. A page window defines the rectangular output area in the output medium (for example, DIN A4 paper page) by specifying the left upper edge of the output area and its width and hight.

When defining a page window, you determine

- which windows appear on a certain page,
- what size the windows have (width, hight),
- their position (distance between the left upper window edge and the left and upper page margins).


Form: Change Page Windows: Z_DEMO_T97
Form Edit Goto Attributes Utilities System Help
Pages Windows Paragraph formats Character formats

Page windows

Page FIRST

Window	Description	Left	Upper	Width	Hght
MAIN	00 Main window	7,00 CH	32,00 LN	71,00 CH	26,00 LN
ADDRESS	Address	7,00 CH	11,00 LN	35,00 CH	10,00 LN
FOOTER	Footer	7,00 CH	61,00 LN	71,00 CH	4,00 LN
HEADER	Header	7,00 CH	1,00 LN	71,00 CH	10,00 LN
SENDER	Sender	43,00 CH	11,00 LN	35,00 CH	15,00 LN

Page window 1 fr. 5

Standard attributes

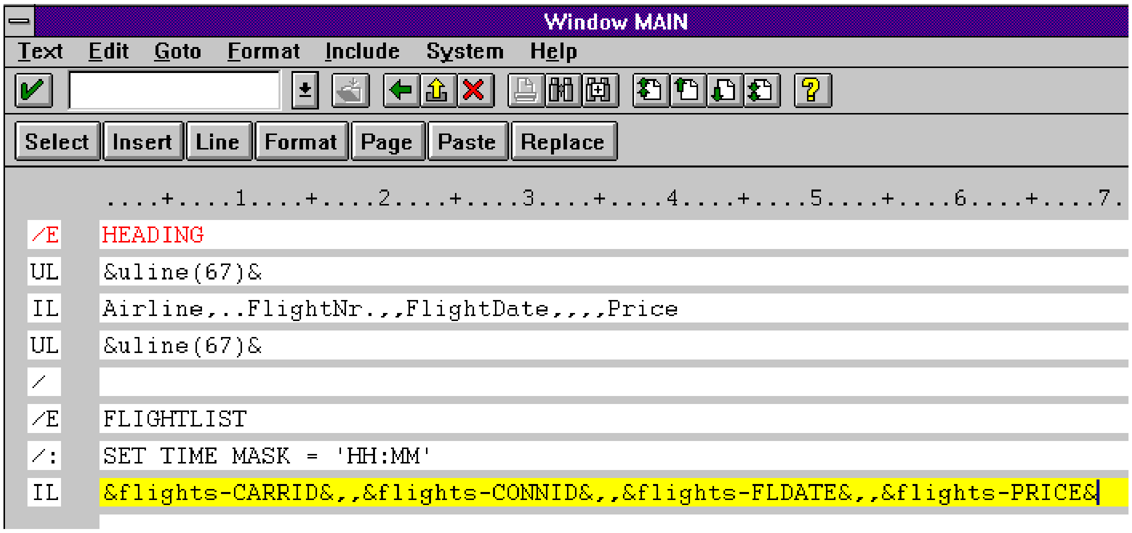
Window	MAIN	Description	Main window
Window type	MAIN		
Left margin	7,00 CH	Window width	71,00 CH
Upper margin	32,00 LN	Window height	26,00 LN

In this example, the user positions the individual windows (previously defined by a user) on the page *FIRST* by specifying the corresponding coordinates. The user can choose between different measurement units the system offers (LN = Lines, CH = Character, and so on).

Text Elements of a Form

SAPscript calls the individual text components of a form **text elements**. To achieve good structuring and readability, you assign a fixed name to each text element in the form. The print program then uses these names to access the elements. This name applies also for translated versions of a text element, while the contents of the text elements depend on the language.

Text elements are related to a window, that is, a print program can call for each window only those text elements that exist in this window. The screen below (shot in the SAPscript line editor) shows the definition of the text elements *HEADING* and *FLIGHTLIST* in the window *MAIN*. The variables used within '&...&' are replaced by the system at output time (see also Representing Text Elements in the PC Editor [Page 20]).



```
      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
/E    HEADING
UL    &uline(67)&
IL    Airline,..FlightNr.,,FlightDate,,,,Price
UL    &uline(67)&
/
/E    FLIGHTLIST
/:    SET TIME MASK = 'HH:MM'
IL    &flights-CARRID&,,&flights-CONNID&,,&flights-FLDATE&,,&flights-PRICE&
```

You can compare text elements with numbered texts in ABAP programs (for example, TEXT-001). However, text elements are much more flexible:

- The length of a text element is unlimited.
- Text elements may contain variable symbols.
- You can use several different formatting options within one text element.
- You can use SAPscript control statements in text elements.

In each window, you may use two different kinds of text elements:

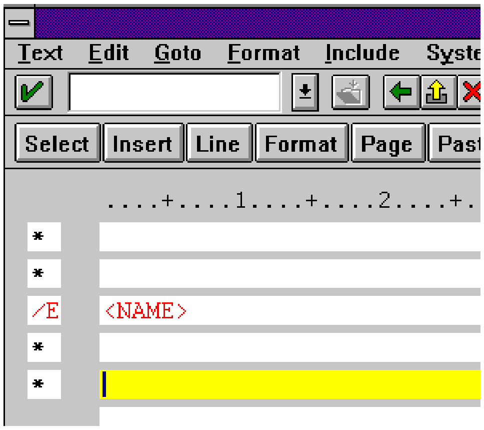
Text Elements with Names [Page 27]

Text Elements Without Names [Page 29]

The print program can Activate Text Elements [Page 30].

Text Elements with Names

You recognize **named text elements** by the paragraph format /E. In the line editor, such a paragraph may look like this:



```
*
*
/E   <NAME>
*
*
```

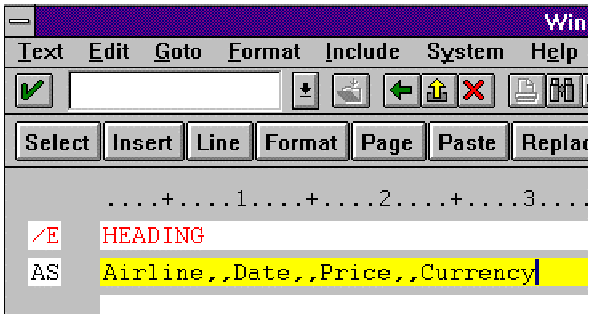
The character string NAME represents the name of the text element. This name can be up to 30 characters long and may consist of letters, digits and/or special characters. The name is followed by the text lines of this text element. The end of a text element definition is marked by the beginning of the next text element (the next /E line).

Note that names of text elements are valid only locally. That is, different windows may contain text elements with the same names.

If you use the PC editor, press the appropriate push button to insert text elements. The system highlights them in a different color (See also: Representing Text Elements in the PC Editor [Page 20]).

To output named text elements, you must use the interface function module WRITE_FORM, which is called in the print program [Page 31].

The example below shows text elements for a column heading within an invoice (AS indicates the standard paragraph format and,, indicates the tab):



```
/E   HEADING
AS   Airline,,Date,,Price,,Currency
```

Text Elements with Names

You can output named text elements only in the windows, in which they are defined. For this reason, you need in the interface function module WRITE_FORM, apart from the name of the text element, the name of the window in which the text element is defined. You call the interface function module from within the print program.

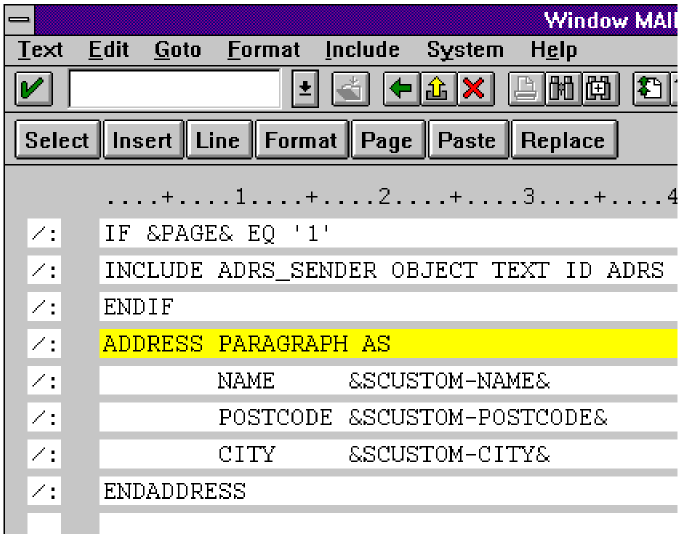
Text Elements Without Names

Nameless text elements or **default text elements** are those text lines and control statements that appear at the beginning of the corresponding window contents without having the /E paragraph format. They include all lines up to the next /E paragraph or to the end of text if no other named element follows. Thus, you can have only one nameless text element in each window.

The differences between named an nameless text elements are:

- The system outputs named text elements only if the print program [Page 31] explicitly calls them in the function module WRITE_FORM. But it outputs nameless text element automatically whenever it processes the corresponding window.
- The nameless text element of the MAIN window appears only once at the beginning of the main window. It does not appear on the subsequent pages of the form which also contain the main window.
- The nameless text elements of all other windows (except MAIN) appear each time.

The following example shows a default text element in the window ADDRESS of a form. The text element outputs the address of a customer, thereby including on the first page the sender short form as include variable for a standard text. To maintain an output-related logic even within the form, SAPscript additionally offers the control statements IF and CASE, which you can use to control the print output depending on the data constellation.



```
      ....+....1....+....2....+....3....+....4
/:    IF &PAGE& EQ '1'
/:    INCLUDE ADRS_SENDER OBJECT TEXT ID ADRS
/:    ENDIF
/:    ADDRESS PARAGRAPH AS
/:              NAME     &SCUSTOM-NAME&
/:              POSTCODE &SCUSTOM-POSTCODE&
/:              CITY     &SCUSTOM-CITY&
/:    ENDADDRESS
```

Activate Text Elements

You activate the text elements of a window from within the print program [Page 31] by setting the parameter FUNCTION when calling the function module WRITE_FORM. FUNCTION may have the following values:

- SET

  Replace all active text elements of the window by the current one (default).

- APPEND

  Append the current text element to the active text elements.

- DELETE

  Delete the current text element from the list of active text elements.

Active text elements are all text elements the system already output to the window when the current page is called.

> ⚠
>
> In the main window, the value *APPEND* within the BODY area equals the value *SET*. The value *DELETE* has no effect in this context. In the areas TOP and BOTTOM, the values act as described above.

Print Program

A print program is an application program (`REPORT or MODULE POOL`) that allows you to print documents to which forms are allocated. The print program retrieves the required data from the database and combines them with the data the user entered. Then, it processes the underlying form, formats the data accordingly, and prints it.

See also: Using Print Programs [Page 32]

Using Print Programs

R/3 applications (FI, CO, MM and so on) deliver standardized forms and print programs that customers may have to adapt to their special needs. Only the close interaction of print program and predefined form allows the user to print forms such as orders or invoices.

One important feature of SAPscript is that forms contain texts with variables besides the layout information. These variables are replaced independent of the formatting and appear as values in the printout. The print program is responsible for retrieving the data from the R/3 system and for the control logic of the output.

This allows the user of SAPscript forms to separate the logic of retrieving data from the layout of the output. The print program retrieves or calculates the required data and determines their output order. SAPscript is responsible for formatting and positioning this data on a print page. Thus, you can modify the layout of the form without having to change the print program.

At runtime of the print program, SAPscript can automatically access data that is defined in the controlling program. Technically speaking: SAPscript retrieves the values directly from the data fields of this program.

Example of a Print Program [Page 33]

Example of a Print Program

The example below shows a typical print program. This simple print program creates an invoice that contains company-related information, date, page numbering, customer address, and all flight bookings of a customer.

For the detailed sample program RSTXEXP1, refer to development class SAPBC460 (corresponding form S_EXAMPLE_1).

Print program: overview

```
* (1) Get customer data
  TABLES: scustom, sbook, spfli.
  DATA: bookings like sbook...
  select * from...
* (2) Open form
  CALL FUNCTION 'OPEN_FORM'
    EXPORTING
      DEVICE = 'PRINTER'
      FORM = 'S_EXAMPLE_1'
      DIALOG = 'X'
    EXCEPTIONS
      others = 1
* (3) Print table heading
  CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      ELEMENT = 'HEADING'
      TYPE = 'TOP'
      WINDOW = 'MAIN'
      FUNCTION = 'SET'
      ...
* (4) Print customer bookings
  LOOP AT bookings WHERE
    CALL FUNCTION 'WRITE_FORM'
      EXPORTING
        ELEMENT = 'BOOKING'
        TYPE = 'BODY'
        WINDOW = 'MAIN'
        ...
  ENDLOOP
* (5) Close form
  CALL FUNCTION 'CLOSE_FORM'
    ...
```

In this example, the first section reads the required data from the database and fills it into internal tables (for example, BOOKINGS). In section (2), the function module OPEN_FORM is called to initialize the print output of the form S_EXAMPLE_1. Then, WRITE_FORM uses the text element [Page 16] HEADING to output general text and the column heading of the invoice in the MAIN window (section (3)). In section (4), the text element BOOKING in the MAIN window is used to output the bookings of a customer that are read in a loop from the internal table BOOKINGS. The address of the customer as well as company-related information is output in other form windows directly, using default text elements. CLOSE_FORM finally ends the printing of the form.

> For each printout of a form, you must use the pair of function modules OPEN_FORM and CLOSE_FORM. You can also use a print program to print several forms, which you can either maintain in separate spool requests or combine into one. In the latter case, you must use the function modules START_FORM and END_FORM.

For more information, see OPEN_FORM [Page 232] and CLOSE_FORM [Page 237] as well as START_FORM [Page 239] and END_FORM [Page 250].

Window Types

When defining a form window [Page 21], you must select a window type for the window. You can choose between three types:

- Constant Windows (CONST) [Page 36]
- Variable Windows (VAR) [Page 37]
- Main Windows (MAIN) [Page 38]

Constant Windows (CONST)

Starting with Release 4.0, the system internally processes windows of type CONST similar to windows of type VAR. Therefore, if you create a new window, always use type VAR.

Variable Windows (VAR)

The contents of variable windows is processed again for each page, on which the window appears. The system outputs only as much text as fits into the window. Text exceeding the window size is truncated; the system does not trigger a page break. Unlike constant windows, the page windows declared as variable windows may have different sizes on different form pages.

As far as the processing of the window contents is concerned, the system currently treats constant and variable windows alike. The only difference is that constant windows have the same size throughout the form.

Main Windows (MAIN)

Each form must have one window of type MAIN. Such a window is called the **main window** of the form. For SAPscript forms, the main window has a central meaning:

- It controls the page break.
- It contains the text body that may cover several pages.
- It allows to fix text elements at the upper and lower margins of the allocated page window (for example, for column headings).

As soon as a window of type MAIN is full, SAPscript automatically triggers a page break and continues to output the remaining text in the main window of the subsequent page. Page windows of type MAIN have the same width throughout the form. The SAPscript composer thus avoids reformatting of the text after each page break.

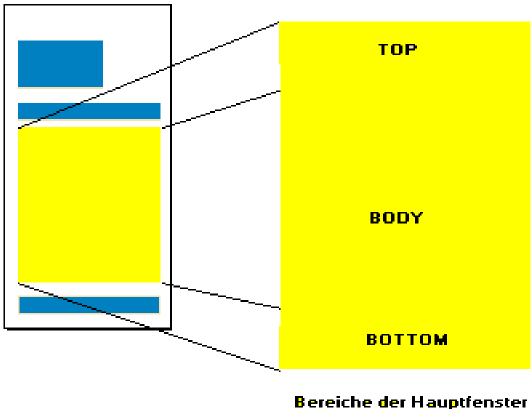
⚠

If a page does not have a main window, the system implicitly processes all other windows of the page and continues with the subsequent page. This page must not call itself as subsequent page (recursive call), since this would produce an endless loop. In such a case, SAPscript terminates the output after three subsequent pages.

For printing header lines or totals, the different output areas [Page 39] of the main window are of special importance.

Output Areas in the Main Window

For outputting texts in the main window, or, more correctly, in a page window [Page 23] of type MAIN, you can choose one of three different areas (see figure below). The upper margin of the main window is called TOP area; the lower margin is called BOTTOM area. The area in-between is called BODY of the main window. The sizes of the TOP and BOTTOM areas depend on the sizes of their text contents. The BODY area varies accordingly.

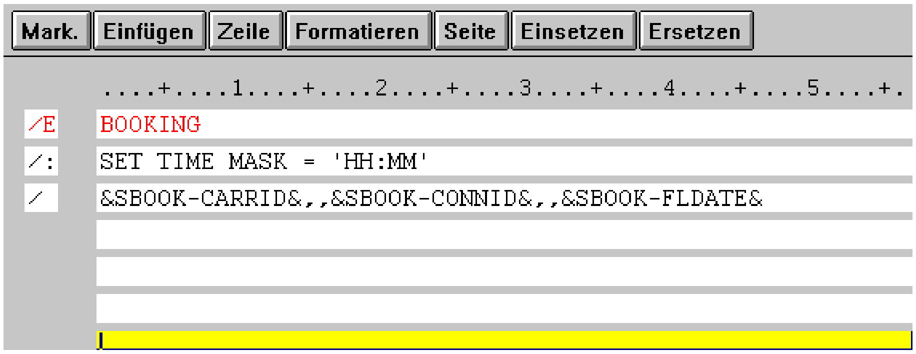


The different areas in the main window

TOP, BODY, and BOTTOM Areas of a Main Window

You can determine or modify the contents of the three areas TOP, BODY, and BOTTOM dynamically during the output of the form. SAPscript automatically outputs these areas on each page of a form that contains a main window. When calling the function modules WRITE_FORM or WRITE_FORM_LINES, the parameter TYPE determines into which of the three areas to output the text. If this parameter is missing in the call, the system positions the output in the BODY area.

A definition of a text element in the main window could look like this:



```
    ....+....1....+....2....+....3....+....4....+....5....+.
/E  BOOKING
/:  SET TIME MASK = 'HH:MM'
/   &SBOOK-CARRID&,,&SBOOK-CONNID&,,&SBOOK-FLDATE&
```

Output Areas in the Main Window

The call in the print program then looks like this:

```
loop at bookings where...

  call function 'WRITE_FORM'
    exporting
      element = 'BOOKING'
      type    = 'BODY'
      window  = 'MAIN'.

endloop.
```

The parameter TYPE of WRITE_FORM has the default value BODY, that is, the system automatically uses this value if the parameter is not explicitly set to another value.

When calling WRITE_FORM, the composer replaces the program symbols in the text element BOOKING (such as &SBOOK-CARRID&) with the contents of the corresponding table fields in the print program [Page 31]. As said before, you can use only fields from Dictionary tables that are defined with TABLES in the print program.

TOP Area [Page 41]

BOTTOM Area [Page 42]

TOP Area

The TOP area always appears at the beginning of the main window. You can use it, for example, to automatically output headings on the subsequent pages for tables that cover several pages.

All output you place into the TOP area using the print program is not formatted at once, but stored internally in SAPscript first. Formatting occurs only as soon as the TOP area in the main window is actually output, that is, as soon as the print program writes text into the BODY area. Variables are retrieved and replaced in that moment as well.

If the BODY area of the main window was filled before the program reached the definition of the TOP area, this definition is used as TOP area for the subsequent page. This means, that you can no longer delete a heading in the TOP area after writing text into the BODY area.

BOTTOM Area

The BOTTOM area appears at the and of the main window. Unlike the TOP area, you can define the BOTTOM area for the current page after the output to the BODY area is complete, provided there is enough space left on the page. Otherwise, the BOTTOM area text is output on the subsequent page.

If the print program outputs text to the BOTTOM area of the main window, it memorizes these lines for subsequent pages (just like TOP lines). At the same time, the composer processes these lines to determine the space the BOTTOM area requires and the space left for the BODY area. At this moment, it replaces the variables set in the BOTTOM area for the current page.

If, during form output, the BOTTOM area is modified, the system reformats the BOTTOM text and adjusts the size of the BOTTOM area. If the remaining space in the page window [Page 23] is not enough, the BOTTOM text is output on the subsequent page of the form. On these subsequent pages, the system always formats the BOTTOM area at the beginning of the main window. This means, that variables appearing in the BOTTOM text always have the value that was valid at the beginning of the main window.

The following table summarizes the points in time, at which the variables in the different windows are replaced with the current values.

Window type	Area	Point of time of variable replacement
MAIN	BODY	immediately
MAIN	TOP	at the beginning of the main window
MAIN	BOTTOM	immediately or at the beginning of the main window of the subsequent page
VAR		after processing the main window
CONST		after processing the main window

Due to these conditions, you cannot use only one variable with changing contents for TOP and BOTTOM areas, since the variable is not always replaced immediately, even if it occurs several times.

⚠

The output of the TOP and BOTTOM area is triggered by the text in the BODY area. Therefore, text elements written to the TOP or BOTTOM areas must not necessarily appear in the output. If the BODY area does not contain any text, output of TOP and BOTTOM elements is suppressed.

How the Composer Works

The composer or form processor is the central formatting module for the print output. It prepares the texts for the different output devices by using the allocated styles or forms.

Processing a form happens in a certain order. You must know some facts concerning the different window types, the setting of subsequent pages, or the dynamic control from within the print program.

Page Control in Forms

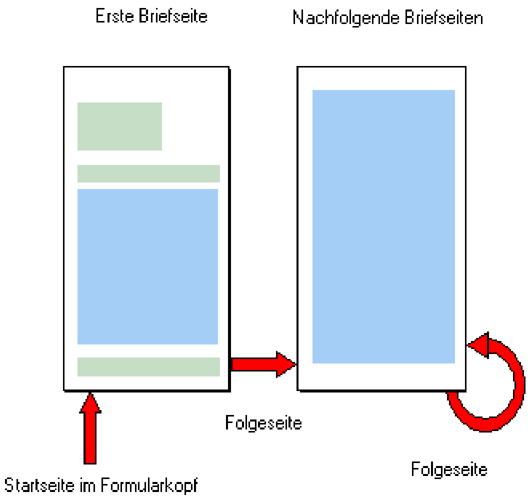
SAPscript automatically triggers a page break as soon as the main window of one page is full. To be able to execute the page break, the system must know on which subsequent page to continue outputting the text. You can specify the subsequent page either **statically** when defining the form, or you can set the subsequent page **dynamically** during form output.

⚠

If the subsequent page is not specified, SAPscript automatically terminates printing, thereby ignoring any other output statements of the application program.

Defining a Subsequent Page Statically

You define the subsequent page statically with the form maintenance transaction. First, specify the start page in the form header [Page 14]. The system automatically calls this page whenever the form is started. With this page, or, more correctly, with the page window of this page, the text output starts. For each page, specify the subsequent page in the page definition. After a page break, the system continues text output on the subsequent page defined for the last page. By specifying start page and subsequent pages, you can define a page sequence.



Static definition of subsequent pages in the form maintenance transaction

Defining a Subsequent Page Dynamically

The page sequence set in the form definition can be changed by the application program dynamically at runtime. If you want the form to start with a page other than the one defined in the form header, specify the desired start page using the parameter STARTPAGE when you call the function module START_FORM. However, this new start page is valid only for the current call of the function module.

If you want to break to a subsequent page other than the one specified in the page definition, use the control statement NEW-PAGE to set the name of the new page.

```
NEW-PAGE <page>.
```

NEW-PAGE ends the output on the current page. The new subsequent page is only valid for the current call of the control statement. You can either include the control statement explicitly into the text of a text element [Page 16] or pass it to the form output using the function module CONTROL_FORM.

Formatting a Form Page

The process of formatting the output is controlled by the text contents in the BODY area of the main window. If the main window is completely filled, or if the control statement NEW-PAGE appears in the main window, the system executes a page break. Only at this point in time the system formats the contents of the windows of the other types and replaces the variables with the current values.

For each other window, the system first outputs the default text element, if it exists. Then it processes and formats the list of the active text elements of this window, which you set using the function module WRITE_FORM with the parameter FUNCTION (SET, APPEND, DELETE). Any text that does not fit into the page window is truncated.

As a consequence to this processing order of the composer, the reservation of space for the TOP and BOTTOM areas must be made beforehand. If the BODY area of the main window already contains text, a new text output to the TOP area does not appear on the current page but on the subsequent page in the TOP area. The same applies for the BOTTOM area. If the BODY area is filled to such an extend that the new BOTTOM text no longer fits into the current main window, this text appears on the subsequent page in the BOTTOM area.

⚠

A frequent error in application programs is that for the subsequent page (for example, NEXT) of a form no main window is defined. If the formatted text of the previous page did not fit into the corresponding main window, the composer searches the subsequent pages for a main window to output the text remainder. However, if the subsequent page of NEXT is NEXT again, the composer encounters an endless loop.

➡

To be able to create correct page breaks in longer text, you should define a main window on each form page.

Form Control

To output SAPscript forms, in the print program you must always start the output with OPEN_FORM and end it with CLOSE_FORM. The function module OPEN_FORM initializes the SAPscript composer and opens the specified form for subsequent output. The system combines all output for this form up to the CLOSE_FORM to one print request. If CLOSE_FORM is missing, nothing will be printed.

To output data in a form, you must use the SAPscript function modules WRITE_FORM, WRITE_FORM_LINES, and CONTROL_FORM. You can use these function modules any number of times in any order between opening and closing a form.

⚠

You cannot use the ABAP statement WRITE to write output to a SAPscript form.

Several Print Requests [Page 49]

Starting a Form Again [Page 50]

Switching Forms [Page 51]

Finding Forms [Page 52]

Several Print Requests

Within one transaction, you can open and close several forms using OPEN_FORM and CLOSE_FORM, however not simultaneously. You can use parameters in the OPEN_FORM to control whether the output is stored in the same print request. But also the SAP spool decides, depending on several plausibility checks, whether new output is appended to an existing print request or whether to create a new print request anyway.

```
CALL FUNCTION 'OPEN_FORM'

CALL FUNCTION ‘CLOSE_FORM’

CALL FUNCTION 'OPEN_FORM'

CALL FUNCTION ‘CLOSE_FORM’
```

➡

You cannot combine ABAP list output and *SAP*script output in one print request.

Starting a Form Again

Usually a print program does not print only one urging letter or one account statement, but several forms for different customers. To have the output for each customer begin with the start page of the form, you must start the current form again and again.

To start a form again, you must first end the current form and then open the form again. Within one print request, first call the function module END_FORM. It executes the final processing for the current form. Then start the form again using START_FORM. Output then begins again on the start page of the desired form.

```
CALL FUNCTION 'OPEN_FORM'
                         :
CALL FUNCTION 'START_FORM'
                         :
CALL FUNCTION 'END_FORM'
                         :
CALL FUNCTION 'START_FORM'
                         :
CALL FUNCTION 'END_FORM'
                         :
CALL FUNCTION CLOSE_FORM
```

➡

If you use START_FORM and END_FORM, you must not specify a form for OPEN_FORM. However, in this case you can use the SAPscript output functions only after opening a form with START_FORM.

Switching Forms

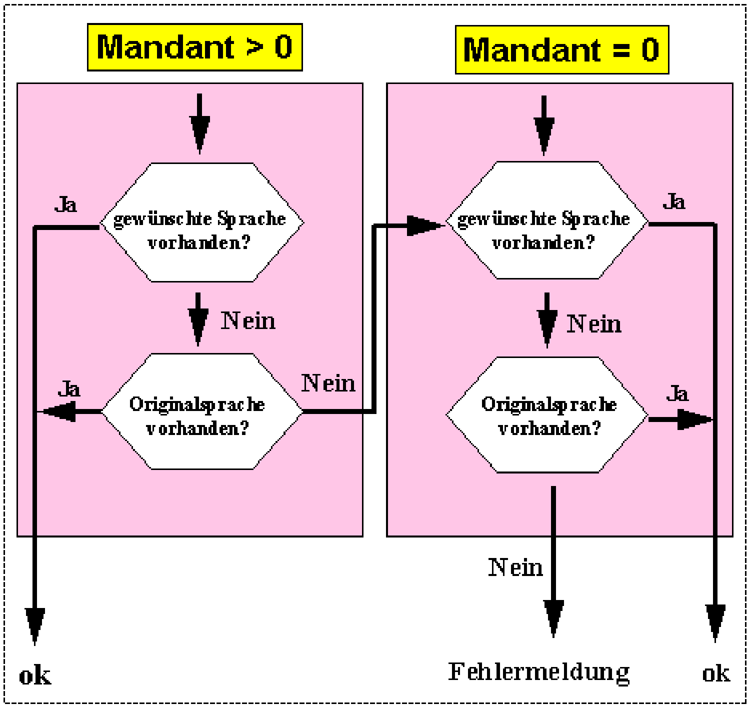
You can switch forms within one print request. This may be necessary if, depending on the recipient, the output language changes or if, depending on the country, a different form layout is required.

You proceed as you would for starting a form again. However, when calling the function module START_FORM, you specify the name of the new form.

> ⚠
>
> When switching forms, make sure that you use only those forms that have the same page format (for example, only DINA4 or only LETTER). However, you can easily mix forms with different page orientations (landscape or portrait format).

Finding Forms

If SAPscript does not find the specified form, it automatically searches for another version of the same form. SAPscript proceeds as follows:


Mandant > 0
Mandant = 0
gewünschte Sprache vorhanden?
Ja
Nein
Originalsprache vorhanden?
Ja
Nein
gewünschte Sprache vorhanden?
Ja
Nein
Originalsprache vorhanden?
Ja
Nein
ok
Fehlermeldung
ok

The SAPscript form processor can use generated forms only. It executes cross-client search only if the current client does not contain a generated version of the form. If during the search it finds a form in another client, it generates the form and stores it in the client from which the search started. If the form is started again, the composer finds it in the current client.

If you use forms delivered by SAP without any modifications, you need not copy these into your production client. SAPscript automatically searches for a form in client 0 if it does not exist in the current client. It then stores the generated version of the form in the production client. If you modify SAP forms, this modification always takes place in a client > 0. Thus, the original SAP version of the form is preserved in the client 0.

Printing Text Lines and Text Elements

Text Lines

To pass the text lines of a text to the output form, use the function module WRITE_FORM_LINES. You must include the text header. However, the SAPscript composer evaluates only the field containing the style (TDSTYLE). If it contains a style, SAPscript uses the layout specifications for paragraph and character formats of this style. If no style is specified, it formats the text according to the paragraph and character formats specified in the form. Paragraph formats that do not exist in the form are replaced by the valid default paragraph formats.

With WRITE_FORM_LINES, you can pass only texts in the SAPscript ITF format. The system does not accept text lines in other formats (field TDTEXTTYPE in the text header > SPACE). In the latter case, the system leaves the function module without further notice.

Text Elements

To call text elements defined in the currently open form, use the function module WRITE_FORM, specifying the name of the desired text element.

Output to the BODY Area of the Main Window [Page 54]

Output to a Window of Type VAR or CONST [Page 55]

Output to the TOP or BOTTOM Areas of the Main Window [Page 56]

Calling Control Statements [Page 57]

Output to the BODY Area of the Main Window

If you omit the parameter WINDOW in the function calls for WRITE_FORM and WRITE_FORM_LINES, the system directs all output to the main window of the form. It immediately formats the the passed text or the desired text element and places them into the output queue. Once formatted output lines are in the output queue, you cannot delete them anymore.

The system ignores any entries for the parameter FUNCTION, that is, evaluates them as APPEND.

Output to a Window of Type VAR or CONST

If you want to write output to another window that the main window, you must use the parameter WINDOW to specify the name of that window. The window need not be defined on the current page.

All other window are processed after the main window. The system first gathers all output directed to them. It stores the text lines to be output in the ITF format; it does not format them at the moment when the function modules WRITE_FORM or WRITE_FORM_LINES are called. Only if the main window triggers a page break does the system format the texts in the other windows and places them in the output queue. This is of special importance when you use variables. They are replaced with the values valid after processing the main window.

The texts gathered for these windows are output again whenever the corresponding window reappears on a subsequent form page. To modify the text contents, you must set the appropriate values of the parameter FUNCTION:

- SET

  Deletes all text lines placed into the window. The system stores the text lines or text element lines passed with the current call for future calls of the window.

- APPEND

  The old window contents remain. New text lines are appended. APPEND is the default value of parameter FUNCTION.

- DELETE

  You can use this function with WRITE_FORM only. It deletes the text element placed into this window using SET or APPEND.

  ⚠

  You cannot delete text lines placed into the window using WRITE_FORM_LINES. To delete these lines, you must first delete the entire window contents using SET, and then write the required texts again.

Output to the TOP or BOTTOM Areas of the Main Window

To write output into the TOP or BOTTOM areas of the main window, set the parameter TYPE to TOP or BOTTOM. You can use this parameter with WRITE_FORM and WRITE_FORM_LINES. As for all output to windows other than MAIN, you can use the parameter FUNCTION to control further text processing.

SET	All lines of the TOP or BOTTOM area are replaced with the text lines or text element lines passed with this call.
APPEND	The old contents of the TOP or BOTTOM area remain; the new text lines are appended. The value APPEND is the default for the parameter FUNCTION.
DELETE	You can use this function with WRITE_FORM only. It deletes the text element placed in the TOP or BOTTOM area using SET or APPEND.

⚠

You cannot delete text lines placed into the window using WRITE_FORM_LINES. To delete these lines, you must first delete the entire TOP or BOTTOM area using SET, and then write the required texts again.

Calling Control Statements

To pass control statements, such as an unconditional page break, use function module CONTROL_FORM. Control statements are always directed to the main window of the form, where the system interprets and executes them immediately.

The Programming Interface

This section explains about the most important structures and interdependencies of the programming interface to word processing. This part of the documentation is designed mainly for developers, administrators, and consultants, who have gathered some knowledge working with SAPscript and want to integrate their own applications according to their requirements.

Structure of Texts [Page 59]

Grouping Texts [Page 64]

Attributes of Texts [Page 67]

Structure of the Text Key [Page 77]

Storing Text Components [Page 80]

SAPscript Data Formats [Page 82]

Authorization Checks [Page 84]

Storing Texts [Page 85]

Text Memory [Page 90]

Work Areas for Texts [Page 96]

The topic below describes the most important steps of integrating SAPscript into an application program, using graphics and examples for better understanding.

This topic is partly based on information given in previous topics. Therefore, it is advisable to read the SAPscript documentation consecutively from the beginning.

SAPscript in Detail [Page 105]

SAPscript Function Modules [Page 160]

Structure of Texts

A SAPscript text module is an object constructed from an administrative information component--the text header--and a table that contains the actual text lines. With many function modules of the SAPscript programming interface, you must specify both components. It makes no difference whether you store the texts with SAPscript or whether you use other text files.

Text Header [Page 60]

Text Lines [Page 61]

Text Header

The text header is a structure. It contains administrative information such as:

- Title of the text module
- Creation data (date, time, creator)
- Change information (date, time, user who made last change)
- Allocated style and/or allocated form
- Text format

Text Lines

The lines of a text are stored in an internal table. Each line consists of a format field and a field with the actual line contents. The contents of these fields is determined by the text format specified in the field TDTEXTTYPE of the text header. Texts may have SAPscript format as well as other formats (for example, WinWord DOC or RTF format).

- **SAPscript Format (ITF Format)**

  A text has SAPscript format if the field TDTEXTTYPE is empty. The texts are stored in the line table exactly as they appear in the SAPscript text editor. This SAPscript format is called ITF format [Page 63]. It consists only of those system characters that can be displayed on the screen and does not contain any hexadecimal codes to control formatting.

  The two-character format field specifies the paragraph type for the subsequent text line. The possible paragraph formats as well as the underlying formatting options are defined in the style or form. Special paragraph formats determine whether the line contents are SAPscript control statements or, for example, comment lines. These paragraph formats are valid for all text modules. The text line field contains either text or a SAPscript statement, if it is a command line.

  The ITF format is no final format; that means, you need the SAPscript composer to create the output format (OTF format) using the layout definitions of the style and the form.

- **Non-SAPscript Formats**

  You can use SAPscript to maintain texts that have a format SAPscript cannot process. The field TDTEXTTYPE in the header of these texts then specifies the respective text format. The text lines of these tests are nevertheless stored in an internal table, which has the structure TLINE [Page 101]. The structure of the individual text lines depends on the respective format. To pass texts in a non-SAPscript format, you can use only the following SAPscript function modules:

READ_TEXT

INIT_TEXT

DELETE_TEXT

COPY_TEXTS

SAVE_TEXT

SELECT_TEXT

RENAME_TEXT

COMMIT_TEXT

EDIT_TEXT

PRINT_TEXT

You cannot mix texts in SAPscript format with texts in another format. This results in restrictions for using texts in other formats:

- You cannot output non-SAPscript texts in SAPscript forms.
- You cannot INCLUDE them in SAPscript texts.
- Variables in the SAPscript syntax are not replaced in these texts.
- All editing functions that depend on an interpretation of the text format cannot be used with these text modules:
  - TEXT_CONTROL_REPLACE
  - TEXT_SYMBOL_REPLACE
  - TEXT_SYMBOL_COLLECT
  - TEXT_SYMBOL_PARSE
  - ...

SAPscript does not interpret the contents of the text line table, but passes the table to the corresponding word processing program, which is designed to process the respective text format. The transfer happens automatically due to the function modules EDIT_TEXT or PRINT_TEXT, which internally call the function modules EDIT_TEXT_FORMAT_xxx or PRINT_TEXT_FORMAT_xxx (xxx = contents of field TDTEXTTYPE). These function modules provide the connection to the corresponding word processing programs.

ITF/OTF Format

The format ITF (Interchange Text Format) is a SAPscript format for storing and displaying SAPscript texts.

This format consists of all characters that can be displayed on the screen (system character set).

Before outputting a text, the SAPscript composer must first convert the ITF format to OTF format (Output Text Format), using the layout definitions of the allocated style and form.

For more information on data formats, see SAPscript Data Formats [Page 82].

Grouping Texts

From a business-oriented point of view, texts are usually related to a certain application. Therefore, most texts are allocated to a certain object, depending on their contents. On one hand, this grouping allows better handling of the texts, on the other hand it facilitates control of internal processes within SAPscript.

Text Object [Page 65]

Text ID [Page 66]

Text Object

In the SAP system, texts do not end in themselves but are usually linked with other business application objects. For example, one text describes a material in detail while another text contains a special agreement concerning an order. Such texts make sense only in connection with the allocated object, since they refer to a certain material or a certain order.

Apart from a contextual relation, these objects also determine certain processing parameters, which the SAPscript functions must consider. For example, when saving a text module, it depends on the object whether the module is directly written to the text database or whether the update task is used.

In the SAPscript environment, these objects are called text objects. Being allocated to a text object is an essential attribute of a text module.

The possible text objects and their respective attributes must be defined in table TTXOB.

Text ID

Usually one text is not enough for an application object. You need several texts to describe all the individual characteristics of an application object. For example, you may need these texts to store information on a customer:

- Sales notes
- Marketing notes
- Accounting notes
- Field service notes

To be able to distinguish between the texts of one object, you need another grouping attribute. SAPscript calls this attribute text ID. You use text IDs to identify the different texts describing the same text object.

The text IDs and their attributes must be defined in table TTXID.

Attributes of Texts

This topic describes the attributes of texts as well as the individual processing steps within SAPscript.

Storage Mode

Text modules are related to an application object. They are created or edited together with the object. You can imagine them as a unit, even though they are stored in different tables. Therefore, the way in which a SAPscript text is written to the database should depend on the way the application object is stored. The following possibilities exist:

- **Direct storage**

  When calling the corresponding save function, the system immediately writes the text module to the text database.

- **Storage in the update task**

  All changes to the text modules of a transaction are stored intermediately in a buffer. Only when the application object is updated does the system write the texts to the log file and, in the update task, stores them in the text file.

- **No storage within SAPscript**

  You can use SAPscript to edit texts that are not stored in the text database. In this case, SAPscript only returns the changed text table to the application program, which is itself responsible for storing the text.

This text attribute is defined in the table TTXOB in field TDSAVEMODE. With some function modules, you can use the parameter SAVEMODE_DIRECT to temporarily switch from storage in update task to direct storage.

Line Width

This attribute determines the line width to be used for this text in the text editor. A SAPscript text line may consist of up to 132 characters.

The editor can display only 72 characters of a text line. If the line width exceeds 72, you can shift the editor display horizontally and thus switch between the left and the right text part. If you set the line width to less than 72, in the editor you can use only as many characters as specified.

Beware that a very small line width may cause problems, since control statements must be contained in one line, and they may be too long then.

You define the line width in table TTXOB in field TDLINESIZE. When initializing the text using the function module INIT_TEXT, the system passes the value you enter here into the corresponding field in the text header. You can modify the line width for each individual text by changing the line size in the text header in field TDLINESIZE after calling the function module READ_TEXT or INIT_TEXT. The system saves the new line width together with the text module.

The line width of the text editor does not effect the line width of a text prepared for printing. This width is determined only from the definition of the paragraph format, the chosen font, and the window width within the form.

Editing Interfaces

If you want to edit a text with the text editor, you can choose between different interfaces. These interfaces determine which functions you can call to edit a text module.

Usually, you work with the interfaces TA or TN:

- *TA* → application texts (variant 1)

  You choose this interface if you can select several texts from the application environment and you want to navigate in this selection list from within the text editor. You need not save each text explicitly, since the navigation functions (*Next text*, *Previous text*, *Back*) automatically save text changes.

- *TN* → application texts (variant 2)

  You choose this interface, if you want to edit only one text from the application environment.

- *TX* → standard texts (text object TEXT)

Depending on the interface, different menu functions are active:

Menu function	**TX**	**TN**	**TA**
Text → Other text	X		
Text → Save	X	X	
Text → Save as	X		
Goto → Next text			X
Goto → Previous text			X
Format → Style → Allocate	X		
Format → Form → Allocate	X		
Format → Convert	X		
Environment	X		

For special applications, there are other interfaces:

- *TD → Documentation*
- *TY → Form texts*
- *TO → Office texts*

These definitions are designed for SAP applications and contain functions which you cannot use in general.

➡

Define the editor interface of a text object in table TTXOB in the field TDAPPL. You cannot change a value specified in this table for an individual text with the SAPscript programming interface.

Editor Title Line

SAPscript displays the following status information in the title line of the editor:

<text ID> <action >: *<text name>* *<text>* Language <*language key*>

The fields have the following meanings:

- *<text ID>*:

  Long text of the text ID from table TTXIT

- <action>:

  The current action is defined by the call of the text editor or by the current processing step within the text editor (change, display, mark, insert).

- *<text name>*:

  Name of the currently edited text.

- *<text>*:

  Additional text from the application program.

- <*language key*>:

  Language ID of the currently edited text.

You can modify this default information.

If the field TDSHOWNAME in table TTXID is empty, the display of the text name is suppressed.

The contents of field *<text>* is supplied by the application program. Its value is passed using the parameter EDITOR_TITLE when the text editor is called.

The application program can even lay out the title line of the editor completely to its own requirements, by using the parameter CONTROL of the function modules EDIT_TEXT or EDIT_TEXT_INLINE. If the parameter field USERTITLE contains an *X*, the title line preset by SAPscript is completely suppressed. The system then displays only the text passed in the parameter EDITOR_TITLE. If you use a variable *&* in this parameter, the system replaces it with the editing function (*Display* <-> *Change*,...).

Text Format

You can use the SAPscript programming interface not only to maintain texts in the SAPscript format ITF [Page 63] but also to pass texts with other formats. When calling the SAPscript function modules, the system passes all text lines, independent of their format, in a line table with the structure TLINE.

However, you can pass texts with other formats only to certain text modules. And the function modules EDIT_TEXT_FORMAT_xxx or PRINT_TEXT_FORMAT_xxx must exist for editing and printing these texts (xxx = text format). These function modules create a connection to the word processing program that can process the specified format.

You can define the text format both in table TTXOB and in table TTXID in the field TDTEXTTYPE, according to the following priority rules:

Table describing the text format

TTXOB-TDTEXTTYPE	TTXID-TDTEXTTYPE	Text format
> SPACE	= SPACE	from table TTXOB
> SPACE	> SPACE	from table TTXID
= SPACE	> SPACE	from table TTXID
= SPACE	= SPACE	SAPscript ITF format

When initializing a text using the function module INIT_TEXT, the system passes the text format to the field TDTEXTTYPE of the text header, depending on how tables TTXOB or TTXID are set and on the priority rules described above.

For an individual text, you can modify the text format by entering the desired format into this field of the text header after the call of function module INIT_TEXT. The system saves the new text format together with the text module.

To modify the text format of an existing text, you must convert the text from the old to the new format, using special converters supplied by the SAPscript programming interface.

Style for Formatting Output

The output format of SAPscript texts is controlled by character and paragraph formats. All output formats you may use for a certain text are combined in a style, which is allocated to the text. The style is stored in the text header in field TDSTYLE.

When creating a new text module using function module INIT_TEXT, the system can automatically store a default style in the text header. However, you must first define default styles for the objects concerned in table TTXOB in field TDSTYLE.

To change the style of a text after executing INIT_TEXT or READ_TEXT, enter the desired style in the text header. If you want to allocate a style for printing only, simply specify the style in the text header before calling the function modules PRINT_TEXT or WRITE_FORM_LINES. With certain text objects, the user can change the style in the text editor, provided the text interface offers the corresponding menu functions. The system then stores the style together with the text.

Form for Formatting Output

A style determines only the paragraph and character formats. To layout the individual print pages, you define forms in SAPscript. You can also specify paragraph and character formats in forms. If no style is allocated to a text, the system uses the paragraph or character formats defined in the form under the same names. If no form is allocated to a text, the system automatically uses the form SYSTEM.

The form is stored in the text header in field TDFORM. When initializing the text using INIT_TEXT, the system copies the default value from the corresponding field in table TTXOB into the text header. After initializing or reading a text, you can change the form by entering another form into the text header. If you want to allocate a form for printing only, simply specify the form in the text header before calling the function module PRINT_TEXT. With certain text objects, the user can change the form in the text editor, provided the text interface offers the corresponding menu functions. The system then stores the form together with the text.

⚠

A form specified in the text header is used only if you format a text directly from within the editor or print it using the function module PRINT_TEXT. If you pass a text to the function module WRITE_FORM_LINES, the system ignores the form specified in the text header and uses the form opened with OPEN_FORM or START_FORM instead.

`INCLUDE` Texts

To include the contents of one text into another, use the statement INCLUDE. The SAPscript composer then includes the second text when processing the first. To specify the text to be included, you must enter the text name with the INCLUDE statement. Entering the text object, the text ID, and the text language is optional. If you omit these entries, the system uses defaults. These defaults are described in the documentation of the SAPscript statement INCLUDE.

The default for the text ID is stored in the field TDID of table TTXID. If the table does not contain an entry here, the system takes the ID of the text that contains the INCLUDE statement.

Structure of the Text Key

SAPscript texts are usually allocated to an object from an SAP application. For example, there are texts on customers, vendors, and materials. To see immediately to which application object a text belongs, the text name should correspond to the respective object name.

	Text name
Customer 0000000012	0000000012
Customer 0000000007	0000000007
Vendor 0000000014	0000000014

As long as the names of customers differ from the names of vendors, no problems can occur. However, since this is not the case, you need another attribute in the text key to make the allocation unique: the text object. The text object links a text directly to the corresponding application object.

	Text object	Text name
Customer 0000000012	KNA1	0000000012
Vendor 0000000012	LNA1	0000000012

In most cases, one text to describe an application object is not enough. For customers, you may need texts for the accounts department, for the marketing department, and for the sales department. To identify these different types of texts, you use the text ID. The text ID, thus, is an attribute for distinguishing texts within one text object.

	Text object	Text name	Text ID
Customer 0000000012 Accounting note	KNA1	0000000012	0002
Customer 0000000012 Marketing note	KNA1	0000000012	0003
Customer 0000000012 Sales note	KNA1	0000000012	0001
Vendor 0000000012 Accounting note	LNA1	0000000014	0001

Since the R/3 system is a multi-lingual system, these different texts, moreover, can appear in different languages. This makes the language ID another integral part of the text key:

	Text object	Text name	Text ID	Text language
Customer 0000000012 Accounting note	KNA1	0000000012	0002	D

Customer 0000000012 Accounting note	KNA1	0000000012	0002	E
Customer 0000000012 Marketing note	KNA1	0000000012	0003	D
Customer 0000000012 Marketing note	KNA1	0000000012	0003	F
Customer 0000000012 Sales note	KNA1	0000000012	0001	E
Vendor 0000000012 Accounting note	LNA1	0000000014	0001	D
Vendor 0000000012 Accounting note	LNA1	0000000014	0001	E

All texts are stored with the respective client. Thus, the client in also part of the text key. The complete text key now consists of the following components:

	Type	Length	Table field for *Like* definition
Client	CLNT	3	THEAD-MANDT
Text object	CHAR	10	THEAD-TDOBJECT
Text name	CHAR	70	THEAD-TDNAME
Text ID	CHAR	4	THEAD-TDID
Text language	LANG	1	THEAD-TDSPRAS

To find out the key of a text, in the SAPscript text editor choose *Goto* → *Header* to display the text header. A dialog box appears that contains the key components of the current text and other information.

The SAPscript function interface checks whether the key fields passed contain valid values:

- Text object:

  The text object specified must be defined in table TTXOB.

- Text ID:

  The text ID specified must be defined in table TTXID together with the specified text object.

- Text language:

  The language specified must be defined in table T002.

- Text name:

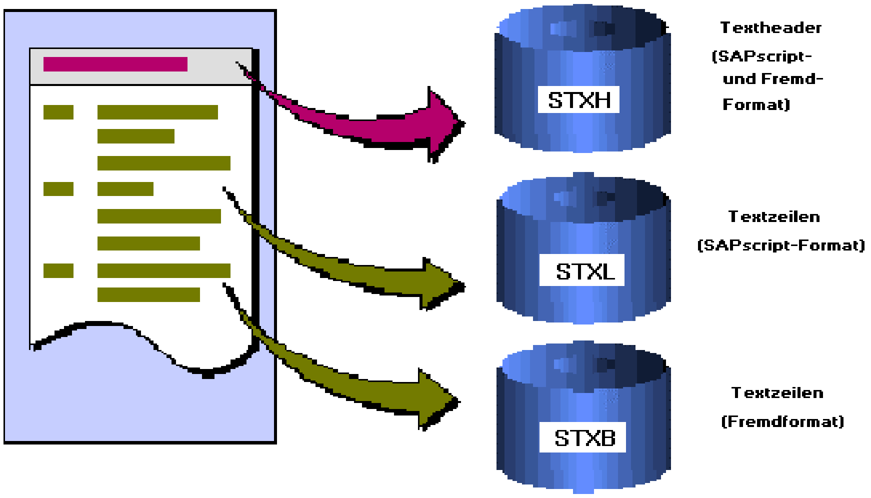
  The text name specified must not contain the characters ‘,’ (comma) and ‘*’ (asterisk).

Storing Text Components

As mentioned before, a SAPscript text consists of a text header and the text lines. If, according to the storage mode, the corresponding text object is stored in a text file, these two text components are stored in separate tables.

> To avoid inconsistencies, use only the corresponding SAPscript function modules to read texts from or write them to text files.

The system stores only those texts in the text file, that have the values ‘D’ (direct) or ‘V’ (update task) specified as storage mode.



STXH: Store text header

The text header is stored in the transparent table STXH. This table contains both the text headers of SAPscript texts and of texts with other formats. When reading the table, the system copies this information into the corresponding fields of structure THEAD, which is the basic structure of the internal work area of the text header.

STXL: Store text lines in ITF format

Table STXL stores the text lines of a text in ITF format [Page 63]. This is a non-transparent table, which can be accessed using IMPORT FROM DATABASE / EXPORT TO DATABASE. The text lines are stored in compressed form.

STXB: Store text lines in other formats

If in field TDTEXTTYPE of the text header a text format appears, the text module has a non-SAPscript format. The lines of such texts are stored in table STXB. This table can also store non-representable characters (HEX codes < SPACE). STXB is a non-transparent table, which can be accessed using IMPORT FROM DATABASE / EXPORT TO DATABASE. The text lines are stored in compressed form.

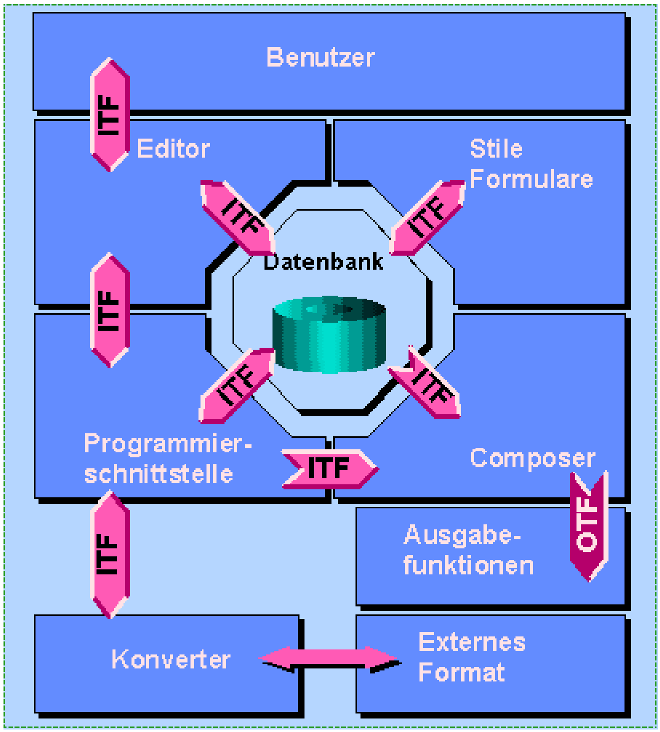
SAPscript Data Formats

Texts created and formatted with SAPscript have a certain data format, the so-called ITF format (Interchange Text Format). It is also used to describe styles and forms.

The ITF format is a readable format, that means, it contains only those characters of the character set that come "behind" the blank character. Characters smaller than the blank character are not used in the ITF format. The ITF format consists of two parts, the format field and the actual line contents. Certain elements of the format are fixed (for example, the paragraph format `'/'` for a new line or `'/:'` to identify the line contents as a control statement). Other elements, such as names of paragraph or character formats can be defined by the user when maintaining the styles and forms.

This format is used in all interfaces between the different components of SAPscript to represent the text lines. In the editor, you edit texts directly in ITF format; the user directly sees the paragraph and character formats and the control statements.

However, this is only a small part of the total extent of the ITF format. Other ITF elements are used to describe styles and forms, even though these attributes are not maintained in ITF format. The table containing the text lines that are passed across the interfaces of SAPscript function modules also contains texts in ITF format.



The composer prepares an ITF text for output, that is converts it into a format that represents the print version. This is the so-called OTF format (Output Text Format). It contains all information on the final line structure and on page breaks. The OTF format is a final format. A text in OTF format can no longer be edited.

The OTF format as the ITF format consists of readable characters. It describes the edited text for a certain output device. Nevertheless, this format is independent of the control language understood by the corresponding output device. The device's print driver converts the OTF format into the language of the output device (for example, Postscript, PCL,...).

Authorization Checks

Authorization Checks for Standard Texts

SAPscript supports authorization checks only for standard texts. These are texts with the text object TEXT, which are edited using transaction SO10. To call this transaction, choose *Tools → Word processing → Standard text.*

The authorization object is S_SCRP_TEXT, with the fields

TEXTNAME	Name of the standard text
TEXTID	ID of the standard text
LANGUAGE	Language key of the standard text
ACTVT	Activity

For TEXTNAME, TEXTID, and LANGUAGE, you can enter single values, intervals, and generic entries, if allowed as authorization values.

For the activity, SAPscript distinguishes between display and change only.

⚠

The authorization for changing a standard text does not automatically imply the authorization for displaying it. If you want a user to both display and change a text, you must allocate authorizations for both activities.

SAPscript executes a create/change or display authorization check when the transaction SO10 is called. To include a standard text into another using *Include → Text → Standard...* in the editor, the user needs only display authorization. The same applies if the user includes standard texts using the control statement INCLUDE. The system executes the check when processing the text module for output.

If the user has no authorization, the system ignores the INCLUDE statement. If an INCLUDE statement in the form specifies a standard text to be included into the output, the system does not execute a check.

To check whether a user has authorizations for standard texts, use the function module CHECK_TEXT_AUTHORITY.

Authorization Checks for Other Texts

For texts allocated to an object other than TEXT, SAPscript does not execute authorization checks. Since these texts are usually allocated to business application objects, SAPscript assumes that the application program checks whether the user is authorized to use the object. If a user is authorized to display a material, this implies the authorization to display the texts allocated to this material. If you want these texts to be independent of the object authorization, you must define new authorization objects for the texts and include an appropriate authorization check call into the application program.

Storing Texts

Modifications to texts should be executed in the same ways as modifications to the application object to which the text is allocated via the text object. This means that text modifications must be stored in the update task whenever the application object uses the update task to store data.

You use the storage mode to determine how a text is stored. You can set the storage mode for each text object used to allocate texts to application objects. The storage modes used by SAPscript are stored in table TTXOB:

- *D* = Store changes directly:

  The system immediately executes all SAPscript functions that change text files.

- *V* = Store changes in update task:

  The system saves changes to text modules internally when the corresponding function is called and only writes them to the text file in the update task, together with changes to the application object.

All other storage modes are not supported by SAPscript. Those texts are not stored in the text file. If you want to store a text that uses such a storage mode using functions from the SAPscript programming interface, the corresponding function module triggers the exception SAVEMODE and stops.

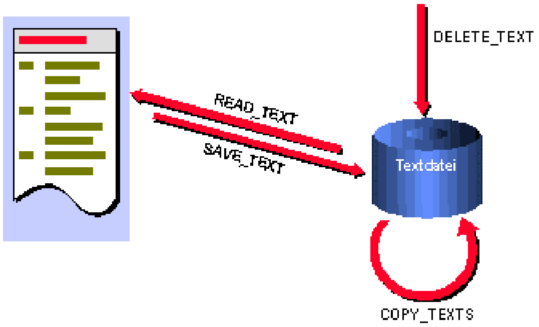
Storing Texts Directly [Page 86]

Storing Texts in Update Task [Page 87]

Renaming Texts [Page 89]

Storing Texts Directly

If a text has the attribute 'direct storage', all changes are written to the text database as soon as SAPscript calls the corresponding function modules (SAVE_TEXT, DELETE_TEXT,...). Since SAPscript does not create backup copies of texts, the old version of the text can no longer be reconstructed.



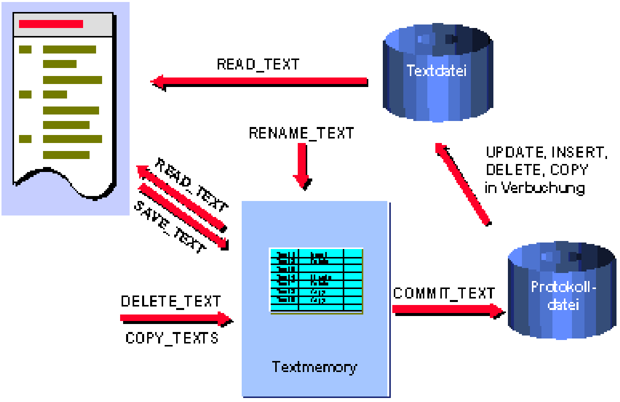
The function module SAVE_TEXT is called implicitly within the function modules EDIT_TEXT and EDIT_TEXT_INLINE. To avoid too many accesses to the text database (performance) or to synchronize the modification time of the texts with those of the application object, you can deactivate automatic storage for these function modules (parameter SAVE). However, you must then call the function module SAVE_TEXT explicitly at the appropriate time. This is usually the time when the other application data is stored as well. This procedure ensures that the system executes all changes to an application object simultaneously.

Storing Texts in Update Task

If the value in the storage mode of a text is set to 'update task', this text is automatically stored. You need no extra function modules for this storage mode. The application program always calls the same SAPscript function modules, independent of the storage mode.

When using storage mode 'V', at the end of the dialog transaction the application program must call a SAPscript function module that prepares the update of the texts processed so far.

Texts maintained with SAPscript consist of a text header and a table containing the text lines. The system keeps these two components in the work areas defined in the application program. The application program must provide these work areas for each text. This means, if two text modules are edited at the same time, two structures must exist to contain the text headers and two tables for the text lines. This is necessary, if several texts are displayed and changed on the same screen. If the texts appear one after the other on different screens, you can reuse the same work areas, provided the old text has been stored. For direct storage, this is no problem, since the changes immediately go to the text file and the system can read the text from there, if the user may need it again during the transaction.



When using the update task, you are not allowed to write the text to the text file, because all database changes are executed only after the user triggered the function 'Update' in the application transaction. For this reason, changes to the text modules are stored intermediately in an administration table controlled by SAPscript: the text memory. If the user wants to edit a changed text again during the transaction, SAPscript fetches the current version of the text from the text memory. The text memory stores the database function to be executed (insert, update or delete) as well as the text header and text lines of the corresponding texts.

At the end of the transaction, the application program must use the function module COMMIT_TEXT to tell the word processing program to pass the texts stored in the text memory to the update task. After this, the text memory is empty again, unless specified otherwise. After all data of the application program are passed to the update task, a COMMIT WORK must occur to update the texts. The function module COMMIT_TEXT itself does not execute a COMMIT WORK.

⚠

You must not change the storage mode set in table TTXOB without adapting the application programs, since, depending on this setting, the programs may have to call the function module COMMIT_TEXT and a COMMIT WORK.

Renaming Texts

The name of a text usually corresponds to the key of the application object. However, in some transactions, this key is determined only at the moment the user executes the function 'Update'. Since texts are edited before this moment, they must have temporary names in this case. These names are then replaced by the finally valid text names shortly before calling COMMIT_TEXT.

To do this, use the function module RENAME_TEXT, which replaces temporary names with the correct text names. You must call this function module before calling COMMIT_TEXT. Otherwise, the texts are stored under the temporary names. You can rename texts in the text memory only.

Text Memory

Within one transaction, the text memory contains all texts with storage mode 'V' that have been processed during this transaction using SAPscript function modules that change the text database.

Structure of the Text Memory [Page 91]

Naming Conventions for the Text Memory [Page 92]

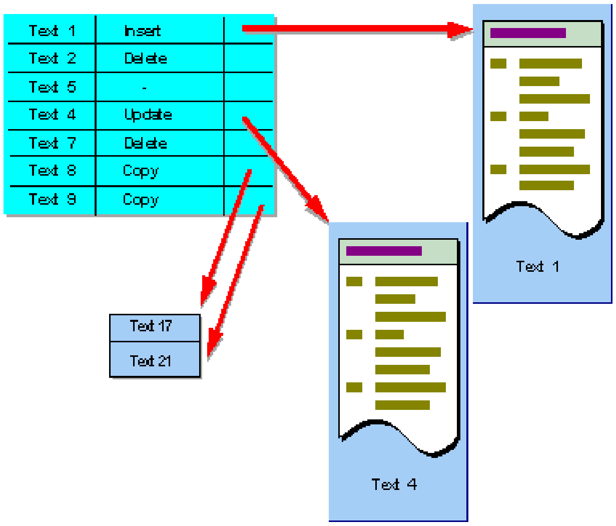
Text Memory and CALL Mode [Page 93]

Keeping Texts in the Text Memory [Page 94]

Changing the Storage Mode Dynamically [Page 95]

Structure of the Text Memory

The text memory consists of an administration table for all texts processed during the transaction. This table specifies for each text which function to execute with it. Depending on this function, other information may be included for that particular text in the text memory.



Text 1	Insert	
Text 2	Delete	
Text 5	-	
Text 4	Update	
Text 7	Delete	
Text 8	Copy	
Text 9	Copy	

Naming Conventions for the Text Memory

SAPscript stores its data in the ABAP memory using different IDs. All IDs for SAPscript entries start with the character string 'SAPLSTXD'.

If you work with the ABAP memory within your application program, you are not allowed to use IDs that start with this character string.

Text Memory and CALL Mode

The text memory is part of the ABAP memory. This means that the system keeps its contents whenever the application program calls a transaction (CALL TRANSACTION, SUBMIT AND RETURN or CALL DIALOG). Apart from transactions, this applies for reports or dialog modules as well. All programs in this CALL hierarchy called by the application program use the same text memory and thus have access to the same texts.

If one of these called programs executes a COMMIT_TEXT, the system transfers all texts to the update task, even those texts that do not belong to that particular program. In such a case, you can use the parameters OBJECT, NAME, ID, and LANGUAGE to specify exactly the texts you want to transfer. You can also use generic names. All other texts, which do not match the selection criteria, remain unchanged in the text memory. If they are changed, you must use other COMMIT_TEXT calls to pass them to the update task.

Keeping Texts in the Text Memory

As described above, the text memory is empty after the final call of the function module COMMIT_TEXT. For a new read access, the system uses the text file. If the update task is slow, the system may read the old text contents.

However, some applications must continue processing texts with the same transaction after a COMMIT_TEXT, for example, to print a document. In this case, you can use the parameter KEEP of the function module COMMIT_TEXT to keep the texts in the text memory. The system flags them to indicate that they have been sent to the update task. If you call COMMIT_TEXT again, the flagged texts are not passed to the update task, unless they have been changed again in the meantime.

Changing the Storage Mode Dynamically

In certain cases, it makes no sense to change a text module using the update task (for example, background programs). In these situations, the system must handle text with storage mode 'V' just like texts with direct storage to the database. The corresponding function modules offer the parameter SAVEMODE_DIRECT, which ensures that a text is stored immediately when calling, for example, SAVE_TEXT. However, you can also use this parameter with COMMIT_TEXT. In this case, all texts are stored in the text memory first and written to the database together at the same time, that is, at the call of COMMT_TEXT. For these texts, you need no COMMIT WORK.

Work Areas for Texts

According to the structure of a text (text header and text lines table), programs that use the SAPscript programming interface must create the appropriate work areas to store these components of a text.

The text header includes all administrative information on a text module. It must be specified with all SAPscript function modules supplied for processing a text module. Its structure is described in the structure THEAD.

The lines table accepts the lines of a text. The structure of a text line is determined by the structure TLINE. All text lines passed to SAPscript using the function module interface must have this line structure.

The specified work areas can contain the information for one text only. If you want to process several texts at the same time, you must create a work area set for each text. After completely processing one text, for example, after saving the text, its work areas can be used for another text. To do this, either use, for example, READ_TEXT to transfer a new text into the work areas, or use INIT_TEXT to initialize them for a new text module.

Text Header : THEAD [Page 97]

Structure TLINE of the Lines Table [Page 101]

Example: Creating Work Areas in the Program [Page 104]

Text Header : THEAD

The text header contains all administrative information on a text module. You must specify it with all SAPscript function modules that exist for processing a text module.

You must create one set of work areas for each text module you want to process using SAPscript. However, you can reuse such a set of work areas after processing of the old text module is complete.

The text header contains the following fields:

TDOBJECT: application object of the text

The field is part of the text key. It contains the name of the text object to which the text is allocated. This object must be defined in table TTXOB.

TDNAME: name of the text

The field is part of the text key. It contains the name of the text module. This name may be up to 70 characters long. An internal structure of the text name is preset by the text object, but not interpreted by SAPscript.

TDID: ID of the text

The field is part of the text key. It contains the name of the text ID to which the text module belongs. The text ID must be defined in table TTXID together with the text object.

TDSPRAS: language key of the text

The field is part of the text key. It contains the language key of the language used to enter the text lines of the text module. The language key must be defined in table T002.

TDTITLE: short description of the text

This field can be used to store a short description of the contents of the text module. To maintain the field, in the editor choose *Goto -> Header*. With standard texts, you can use the search function to find texts that contain certain character strings in this field.

TDSTYLE: style including paragraph and character formats

If a style is allocated to a text module, this field contains the name of the style. The system then edits paragraphs according to the definitions in this style. If no style is specified, the system uses the corresponding information from the form into which the text module is output.

TDFORM: form for output

If a form is allocated to a text module, this field contains the name of the form. The system then uses the formatting information of the form to format the text for output. However, the form specified here is used only, if the text is output using the function module PRINT_TEXT.

TDVERSION: version number of the text

The field contains the version number of the text. When creating the text, the system sets the number to 1. For each change to the text, the system increases the number by 1. If you delete a text module and create a new one using the same name, the value is reset to 1.

TDFUSER: name of the user who created the text

This field contains the name of the user who created the text module. The system retrieves the user name from the system field SYST-UNAME. The field contents remain the same, even if the text module is changed. If you delete a text module and create a new one using the same name, the system enters the currently active user into the field.

TDFRELES: release at which the text was created

This field contains the release of the SAP system at which the text module was created. The system retrieves the value from the system field SYST-SAPRL. The field contents remain the same, even if the text module is changed. If you delete the text module and create a new one using the same name, the system enters the current SAP release into this field.

TDFDATE: creation date

The field contains the complete date of the day on which the text module was created. The system retrieves the value from the system field SYST-DATUM. The field contents remain the same, even if the text module is changed. If you delete the text module and create a new one using the same name, the system enters the current date of day as creation date.

TDFTIME: creation time

The field contains the time of day at which the text module was created. The system retrieves the value from the system field SYST-UZEIT. The field contents remain the same, even if the text module is changed. If you delete the text module and create a new one using the same name, the system enters the current time of day as creation time.

TDLUSER: name of the user who last changed the text

The field contains the name of the user who last changed and saved the text module. The system retrieves the value from the system field SYST-UNAME. If the text module is new, this field contains the same value as TDFUSER.

TDLRELES: release at which the last change to the text occurred

The field contains the release of the SAP system at which the last change to the text module took place. The system retrieves the value from the system field SYST-SAPRL. If the text module is new, this field contains the same value as TDFRELES.

TDLDATE: date of last change

The field contains the complete date of the day on which the text module was last changed and saved. The system retrieves the value from the system field SYST-DATUM. If the text module is new, this field contains the same value as TDFDATE.

TDLTIME: time of last change

The field contains the complete time of day at which the text module was last changed and saved. The system retrieves the value from the system field SYST-UZEIT. If the text module is new, this field contains the same value as TDFTIME.

TDLINESIZE: line width in the editor

The field contains the line width of the text module. This is the line width for formatting the text in ITF format [Page 63], that is, the format used for display in the editor. The lines of a paragraph are formatted in the width specified here, however, no wider than 72 characters. Exceptions are lines with paragraph formats that exclude the lines from formatting (command lines, long text lines, raw lines...). This line width has nothing to do with the line width when formatting a text for print output. The latter is determined individually for each paragraph by settings in the form and in the style.

The line width must have a value between 40 and 132.

TDTXTLINES: number of text lines

The field contains the number of text lines of this text module, which are stored in the corresponding lines table.

TDOSPRAS: original language (only for forms and styles)

This field is interpreted for forms (object = FORM) and styles (object =STYLE) only. Forms and styles consist of a definition part and a text part. The definition part is language-independent and occurs only once, whereas the text part can occur in several languages. This field specifies which of the existing languages is the original language, that is, the basis for translation. The definition part is also stored as text module and must, therefore, have a language key even though it is language-independent. The original language thus also determines the language key of the definition part of a style or form.

TDTRANSTAT: translation status (only for forms and styles)

Text Header : THEAD

TDMACODE1: short title 1

The field should contain a short title in the form of a key word. When searching for standard texts, the selection can be limited by entering a search character string for this field.

TDMACODE2: short title 2

The field should contain a short title in the form of a key word. When searching for standard texts, the selection can be limited by entering a search character string for this field.

TDREFOBJ: object of the reference text

The field contains the object name of the reference text to which the current text module refers.

TDREFNAME: name of the reference text

The field contains the name of the reference text to which the current text module refers.

TDREFID: text ID of the reference text

The field contains the name of the text ID of the reference text to which the current text module refers.

TDTEXTTYPE: text format

Texts can be stored in different text formats. This field contains the name of the format of the text module. If the field is empty, SAPscript assumes that the text lines have ITF format and calls the SAPscript text editor. If the field contains a value, the system assumes a non-SAPscript format and calls the word processing program competent for this format.

TDOCLASS: object class

To allocate a text object to an application class, you can specify the name of a program class in this field. The value must be defined in table TRCL. SAPscript uses this field only for maintaining forms.

TDHYPHENAT: not used

TDCOMPRESS: not used

Structure TLINE of the Lines Table

The lines table contains the lines of a text module. Its structure is determined by the structure TLINE. All text lines passed to SAPscript across the interfaces of function modules must have this line structure.

SAPscript has its own format for text lines, the ITF format [Page 63]. A text line in this format consists of two fields, the format field and the actual line contents. These two fields form the structure TLINE. The ITF format also determines the meaning and the typing of the control information allowed in the format and line fields.

You can use SAPscript to administer texts that have other formats that ITF. The information on the text format is stored in the text header. To pass texts with other formats to SAPscript via function modules, you must also use the structure TLINE. SAPscript, however, does not interpret the contents of the text lines but passes them on to the function modules, which call the interface to the external word processing program.

TDFORMAT: format field

The format column contains format keys that determine the output formatting of the text or start control statements. A format key determines the beginning of a new text paragraph and its formatting. All subsequent text lines, which contain blanks in this field, belong to the same paragraph. These lines are treated as texts with automatic line feed. The SAPscript editor formats these lines by fitting as many words as possible into one editor line, always considering blanks between words.

The formatting in the editor is independent of the formatting for output. Output formatting of a text is visible only when the text is actually printed or displayed on the screen. For output editing, the layout specifications of the paragraph format are evaluated. The possible format keys and their meanings are determined in styles or forms.

If you allocate a style or form to a text module, you can use the paragraph formats specified there for the text layout. Format keys which the user can define consist of one or two characters. The letters from A to Z and the numbers from 0 to 9 are allowed. The paragraph format must begin with a letter. If a format key is not part of the allocated style or form, the system uses the default paragraph of that style or form.

Some format keys are predefined by SAPscript. You can use them in all texts:

`*` default paragraph

For output formatting of the subsequent paragraph, the system uses the formatting specifications that correspond to the paragraph defined as default paragraph in the allocated style or form.

`/` new line

For output formatting, the subsequent text appears on a new line. The formatting specifications of the last paragraph format apply.

`/:` command line

The system interprets the characters in the actual text line as control statements rather than text. Control statements are interpreted and executed when the test is formatted for output. The entire control statement must fit into one line; spreading it over subsequent lines is not allowed. The SAPscript editor does not format control statement lines.

`/*` comment line

When formatting a text for output, the system does not output this line.

`=` long line

This line is not subjected to line formatting in the SAPscript editor. The system appends the text in this line directly to the last character of the previous line. If you want some space in the output between the first and the second line, you must start the long line with at least one blank.

`/=` long line with line feed

This line is treated just as = (long line), but when formatting for output, the subsequent text appears in a new line.

`(` raw line

The SAPscript composer does not interpret the subsequent line when formatting the text for output. This means that any character formats, variables, tabs, mask characters, or hypertext links contained in this line are not evaluated and reach the output device unchanged. In addition, the text in this line is directly appended to the last character of the previous text line. If you want some space in the output between the first and the second line, you must start the raw line with at least one blank.

`/(` raw line with line feed

This line is treated just as ((raw line), but when formatting for output, the subsequent text appears on a new line.

`>x` fix line

This line in input-disabled in the SAPscript editor. You cannot delete or split it. You can create a fix line only from within the program, for example, to allocate a fixed structure to a text, which the user cannot change. Replace the ‘x’ with any number or any letter to distinguish, for example, between different subtitles.

If several fix lines with the same identifier appear in sequence, the SAPscript editor considers them as a unit. You cannot insert anything in-between in the editor. For output formatting, SAPscript interprets the first two characters of the line contents of a fix line as paragraph format. Either specify the desired paragraph format in these two characters or leave them blank.

TDLINE: text line

The field TDLINE contains the actual text. Depending on the format field of the line, the system interprets the line contents as

- text
- control statement
- comment

The control statement contained in a SAPscript text line must consist of readable characters. You cannot use hexadecimal codes, which cannot be displayed on the screen. If you do use hexadecimal codes, using SAPscript function modules may present unwanted results.

Beside the actual text, text lines can contain character formats and variables.

Character formats define the formatting of individual characters or character strings within a paragraph. They begin with the escape symbol <z> and end with the characters </>. If you leave out the end sequence, the system uses the character format until the end of the paragraph. You can nest character formats.

‘z’ is the name of the character format, which is defined either in the allocated style or form. Character formats defined by the user consist of one or two characters. Allowed are the letters from A to Z and the numbers from 0 to 9. The name must start with a letter. If a character format appears in the text that is defined neither in the style nor in the form, SAPscript ignores it.

Apart from the character formats the user can define, there are some formats predefined by SAPscript, which can be used in all texts:

<(>... <)> raw character

The characters in-between this character format are output unchanged. This allows you to pass certain character sequences and SAPscript variables to the output. This character format corresponds to the paragraph format 'raw line'.

`<x>` special character

Use this format to output a character that you cannot enter via the keyboard. Replace ‘x’ with the number of the SAP character. All characters valid in the SAP system have a unique number. However, you can print or display this character only, if it is defined in the system character set and if the character set of the corresponding output device also contains it.

Variables (or symbols) are placeholders for values that you set at the moment of actually formatting the text for output. To recognize variables, they must have a certain structure:

- Variables must be included in & characters.
- The variable name must not contain blanks.
- The entire variable must fit into the field TDLINE.

Example: Creating Work Areas in the Program

To define the work area for the text header, use a structure like THEAD:

```
DATA <name of textheader> LIKE THEAD.
```

To define the work area of the lines table, use an internal table with the line structure TLINE:

```
DATA <name of linetable> LIKE TLINE OCCURS <n> WITH HEADER LINE.
```

You can use the above short forms for declaring work areas only, if you use release 3.0 or higher. They have the same effects as

```
DATA BEGIN OF <name of textheader>.
  INCLUDE STRUCTURE THEAD.
DATA END OF <name of textheader>.
```

or

```
DATA BEGIN OF <name of linetable> OCCURS <n>.
  INCLUDE STRUCTURE TLINE.
DATA END OF <name of linetable>.
```

SAPscript in Detail

Steuerung der Druckausgabe [Page 134]

This section offers further details on some SAPscript areas. Besides a list of the control tables used, more information on how to control the printout and the editor is provided.

Integrating Text-Processing into Application Programs [Page 106]

SAPscript Control Tables and Structures [Page 126]

Controlling Print Output

Editor Control [Page 155]

Integrating Text-Processing into Application Programs

This section deals with the explicit integration of text-processing into application programs. Examples are used to clarify the explanations. Special tips may help you with problems.

Reading Texts

You can process texts only if they are stored in the internal work areas of the program. Therefore, you must first transfer a text into the work areas.

To transfer the text header of a text into the specified structure and the text lines into the specified lines table, use the function module READ_TEXT.

Usually, the system reads a text from the text file. However, for texts with storage in the update task, the system first looks into the text memory to see whether it contains a currently processed version. If so, the system reads this version of the text into the work area, otherwise the text version from the text file.

To read a text version stored in the archive, use the parameter ARCHIVE_HANDLE.

If the desired text does not exist, READ_TEXT ends with the exception NOT_FOUND. The contents of the work areas for text header and text lines are then undefined. To be able to use these work areas for another text, you must first initialize them with INIT_TEXT.

The function module READ_TEXT also handles text references. It reads the reference chain to its end and supplies the text lines of this text in the lines table as well.

Saving Texts

To re-transfer texts from the internal work areas to the text file, use the function module SAVE_TEXT.

The application program does not know whether the text is new or a changed version of an existing text. To be able to find this out, the program must read the text file first.

If a text exists, the transferred text lines overwrite the old version. If it does not exist, the system creates it. If you know from the beginning that the text is new, you can suppress this read process using the parameter INSERT and thus improve performance.

A text you want to store in the text file must consist of at least one line whose paragraph format or line contents is unequal to SPACE. Otherwise the system automatically deletes this text from the text file.

Changes to the text file are valid at once if the text object of the text is set to direct storage. If it is set to storage in update task, the text changes are temporarily stored in the text memory. The function module COMMIT_WORK then transfers them to the log file, from where they are updated with the next COMMIT WORK.

The function module SAVE_TEXT can handle only texts that are eventually stored in the text file, that is, text with storage mode 'D' or 'V'.

Deleting Texts

To explicitly delete texts from the text file, use DELETE_TEXT and specify the key of the desired text. You must not read the text first.

With DELETE_TEXT, you can specify the text name, text ID, and text language generically as well. This allows you to delete, for example, all texts belonging to an application object, in one call.

Delete all texts with company code 0001 for customer 4711.

```
CALL FUNCTION  'DELETE_TEXT'
  EXPORTING  OBJECT    = 'KNB1'
             NAME      = '00000047110001'
             ID        = '*'
             LANGUAGE = '*'.
```

Delete all company code-related texts of customer 4711.

```
CALL FUNCTION  'DELETE_TEXT'
  EXPORTING  OBJECT    = 'KNB1'
             NAME      = '0000004711*'
             ID        = '*'
             LANGUAGE = '*'.
```

The texts are deleted from the text file immediately, if the text object of the text(s) is set to direct storage. If it is set to storage in update task, the deletion request is stored in the text memory first. The function module COMMIT_WORK then transfers the request to the log file, from where it is executed with the next COMMIT WORK.

The function module DELETE_TEXT can handle only texts that are stored in the text file, that is, text with storage mode 'D' or 'V'.

Calling the Editor

Use the function module EDIT_TEXT to branch to a (fullscreen) text editor that allows you to edit the transferred text. Depending on the text format in the text header (TDTEXTTYPE), the system calls the corresponding editor. If the format field is empty, it calls the SAPscript editor. Otherwise it internally calls the function module EDIT_TEXT_FORMAT_xxx, which is responsible for establishing the link to the text editor that can process texts in the format xxx.

The SAPscript editor offers several functions.

The editor interface, that is, the activated functions in the editor menus, are determined by the text interface allocated to the text object in table TTXOB. In addition, you can set certain attributes of the interface using the parameters CONTROL, DISPLAY, and EDITOR_TITLE when calling the function module EDIT_TEXT.

Call the Editor Without Navigation

If you want to edit only one text in the editor, you should specify the interface TN for the corresponding text object in table TTXOB. After editing the text, you leave the editor and return to the application screen.

Call the Editor with Navigation

If you want to select several texts from the application screen and must, therefore, navigate in the text editor in this selection list, you must use the interface TA. This interface offers the functions *Next text* and *Previous text* in the *Goto* menu. You can use them to move among the selected texts without having to leave the editor.

From within the program, you must call the function module in a loop. After editing one text, you end the function module and return to the calling program. The program decides, depending on the parameter RESULT, field USEREXIT, which function the user used to leave the text editor, and then, depending on that function, either calls the text editor with another text or leaves the loop.

How to proceed:

1. Provide the desired text of the selection list by placing the text header and the text lines into the appropriate work areas.

2. Use the parameter CONTROL to specify which function *Next text* or *Previous text* you want to activate:

   `APP_NEXT = 'X'`, if the text passed for the subsequent editor call is not the last text in the selection list,

   `APP_PREV = 'X'`, if the text passed for the subsequent editor call is not the first in the selection list.

3. Call the text editor.

4. Check the return parameter RESULT, field USEREXIT to determine the next step:

   – Go to the beginning of the loop if the value of the field is either `'N'` (Next text) or `'P'` (Previous text).

   – Otherwise leave the loop.

Finding Texts

This topic tells you how to find out which texts belong to an application object.

The name of SAPscript texts (field TDNAME) should correspond to the key of the application object to which the text is allocated. Different text types among the texts of an application object are determined by the text ID (field TDID) and by the language of the text.

If the text name corresponds to the key of the application object, you can identify all texts belonging to that object by the key. You need not store fields in the data record of the application object that contain the text name.

If an application transaction wants to know which texts exist for a material or a customer, it calls the function module SELECT_TEXT to find out. The function module in a result table returns the headers of all texts that match the selection criteria.

The function module selects texts not only from the text database, but also from the text memory.

You want to find all texts that belong to customer 4711 (centrally). The allocated text object is KNA1. The system returns the text headers of the found texts in table CUSTOMER_TEXTS:

```
DATA: CUSTOMER_TEXTS LIKE THEAD OCCURS 10.

CALL FUNCTION 'SELECT_TEXT'

  EXPORTING OBJECT     = 'KNA1'

            NAME       = '0000004711'

            ID         = '*'

            LANGUAGE   = '*'

  TABLES    SELECTIONS = CUSTOMER_TEXTS.
```

If you want to search for the texts of a customer in the company code 0001, use the following parameters of the function module SELECT_TEXT:

```
CALL FUNCTION 'SELECT_TEXT'

  EXPORTING OBJECT     = 'KNB1'

            NAME       = '00000047110001'

            ID         = '*'

            LANGUAGE   = '*'

  TABLES    SELECTIONS = CUSTOMER_TEXTS.
```

If you do not know the structure of the text name, use the corresponding application transaction to display one of the texts in the text editor and request information on the text by choosing *Goto → Header*. The text name of this text appears, from which you can now easily determine the text name structure.

Copying Texts

To copy SAPscript text, proceed like this:

1. Read the text you want to copy using READ_TEXT.
2. Rename the text by entering the new values into the fields TDOBJECT, TDNAME, TDID in the text header.
3. Save the text using SAVE_TEXT.

Use this procedure if the text to be copied is a model only and will be modified before saving, either from within the program or by the user in the text editor.

The disadvantage of this procedure is that the entire text is read into the internal work areas and then rewritten to the text database, possibly without any changes. The performance of this procedure is bad, especially so, if you want to copy a lot of texts in one transaction and store them using the update task.

In such a case, you better use the function module COPY_TEXTS: Enter the key of the texts to be copies and their new names into a table. The system then copies the texts by copying only the required data records.

Inserting Text Lines into Application Screens

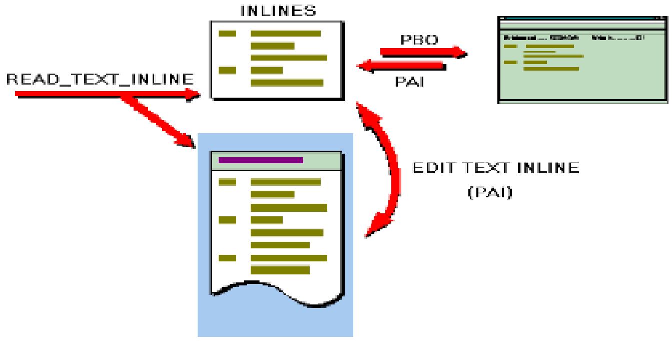
In many cases, it is superfluous to call the SAPscript fullscreen editor for editing transaction texts. It may be sufficient to offer only the first text lines on the screen. This allows you to display several texts on one screen, even together with other data of the application object. If all text lines fit into this reserved area, the user can even change the text directly on the application screen. You must call the editor only if you allow the user to enter more than one line.

If a text contains more lines than are reserved on the application screen, you can display a flag, The user then knows that the text displayed on the screen is not complete.

This procedure is called inline processing. SAPscript supports it only in parts. For inline processing, you use the function modules READ_TEXT_INLINE and EDIT_TEXT_INLINE, which have the same functionality as READ_TEXT and EDIT_TEXT.

READ_TEXT_INLINE transfers the first few text lines into a second internal table (INLINES). The application program must then transfer these lines at the event PBO into the corresponding screen fields. The application must decide whether to display the character formats on the inline screen. If not, it may be necessary for the application program to include a paragraph format. At the PAI event, the application program must retransfer the corresponding screen fields into the table INLINES and call the function module EDIT_TEXT_INLINE. It merges the text lines in table INLINES with the text lines in table LINES, reformats the text, and places the first few text lines back into table INLINES.

To automatically branch to the fullscreen editor after merging the texts, use another parameter. To determine, whether a text consists of more lines that fit onto the inline screen, and to display the corresponding flag to the user, compare the number of lines of table LINES with the number of lines of table INLINES.



In contrast to editing texts in the fullscreen editor, inline processing encounters some restrictions:

- The system cannot automatically concatenate words that are split by the end of the line.
- Inline processing does not support the editing functions offered by the fullscreen editor. You can use only the elementary editing functions for screen fields, which are generally available.
- You cannot scroll the text.

  You only see the beginning of the text lines. If they are wider than the screen fields, you must use the fullscreen editor to edit the hidden part of a line.

- If you suppress the format field of the SAPscript text line in the display, you must use the fullscreen editor to enter control statements.
- The system stores changes to the text lines immediately in the original lines table of the text, automatically using function *Save* of the editor. Thus, you cannot restore the original version of the text, unless you cancel the update task and reset all changes made by the entire transaction.

Inserting Other Texts

SAPscript supports two procedure to insert texts from other text modules into a text, depending on the purpose behind it.

Including Texts [Page 118]

Referring to Texts [Page 119]

Including Texts

To include a text into any text module, the user enters the INCLUDE control statement in the SAPscript editor. When formatting the text for print output, the system reads the text lines of the specified text and inserts them in the current text printout.

In each text, several INLCUDE statements can appear together with other text lines. In the SAPscript editor, you can see only the INCLUDE statement, not the lines of the text you include. With an INCLUDE statement, you always include all lines of the specified text. You can nest INCLUDE statements.

If you change a text, these changes effect all texts that include it. If you delete a text, a corresponding INCLUDE statement is without effect.

To change text lines of an included text, you must resolve the INCLUDE statement. In the text editor, choose *Edit → Selected area → Expand INCLUDE.* The system now copies the text lines into the current text and deletes the INCLUDE statement. Afterwards, there is no more link to the text originally specified in the INCLUDE statement. This means, that changes to the original text no longer effect the current text.

You cannot INCLUDE any texts, but are restricted to texts of certain text objects and text IDs, depending on the text environment the INCLUDE statement appears in. When including a standard text, SAPscript checks whether the user has the display authorization for this text.

For more information, see the documentation *Style and Form Maintenance* under the description of the INCLUDE statement.

Referring to Texts

You use text references, if you also want to refer to the allocated application object. You use this procedure, for example, when creating an order and a similar order already exists, part of whose data you can reuse.

You refer to texts from within the program; that is, to establish a text reference the application program must call the SAPscript function module REFER_TEXT. A text that refers to another text module may itself not contain any further text lines. The reference is established by storing the referred text in the fields TDREFOBJ, TDREFNAME, and TDREFID of the text header of the current text. The system automatically sets the language of the referred text to the language of the current text.

For text references, the system stores only the text header. The text lines are filled when the system reads the reference text. A reference can extend over several levels, that is, a referred text can itself refer to another text. In this case, when reading the texts, the system works through the entire reference chain and uses the text lines of the text at the end of the chain.

For text references, there are no restrictions concerning the text object, text ID or text name. The application program that establishes the reference must make sure that it refers to the correct texts. SAPscript does not execute an authorization check.

As for including texts, for printing or displaying the reference text in the editor the system always uses the current text version. In contrast to INCLUDE texts, the SAPscript editor displays all lines of the referred text. However, the text area in the editor is input-disabled, so that you can neither change the text nor insert other text lines. If you want to change the text, you must first unlock it. In the editor choose *Text* → *Unlock*. The system then resolves the reference and copies the lines, as for including, into the current text module. The text lines of the editor are now input-enabled, but there is no more link to the original reference text.

If you want to resolve a reference from within the program, you must first make sure that the text lines of the reference text are stored in the corresponding lines table. Then simply delete the fields TDREFOBJ, TDREFNAME, and TDREFID in the text header.

If a reference text no longer exists, the system ends the corresponding function module, for example, READ_TEXT with the exception REFERENCE_CHECK. The SAPscript composer ignores missing reference texts when formatting a text for printing.

Function modules:

REFER_TEXT

READ_REFERENCE_LINES

READ_TEXT

Processing Texts from Within Programs

SAPscript texts are stored in the lines table in ITF format. This format consists of readable characters only and does not contain any hexadecimal codes < SPACE.

To process the lines table from within an application program, use the ABAP statements for table processing (LOOP, READ, INSERT, DELETE). Since the fields of structure TLINE are of type C, you can use all ABAP statements available for processing type C fields.

SAPscript offers a number of function modules to modify the text lines table, taking into account the syntax and semantics of the SAPscript ITF format.

Converting SAPscript Texts

SAP*script* texts are stored in Interchange Text Format (**ITF**). As interface to other word processing programs, SAP*script* offers conversion programs to the text file formats Rich Text Format (**RTF**) and **ASCII** as well as migration for R/2 texts.

RTF Files

RTF files contain the entire formatting information of a text and can be interpreted as well as created by all common word processing programs.

RTF files created from SAP*script* texts largely contain the character and paragraph formats from the style or form of the original text. They can easily be read into modern word processing programs; however, for using older DOS programs you must make some preparations first: For Microsoft Word (starting with version 5.0), for example, you must convert the RTF file on operating system level into a Word text file and a template using the following commands:

MS-DOS	rtf_dos file1.rtf file2.txt file3.dfv /o/c
MS-DOS	rtf_dos file1.rtf file2.txt file3.dfv /o/m
OS/2	rtf_os2 file1.rtf file2.txt file3.dfv /o/c
OS/2	rtf_os2 file1.rtf file2.txt file3.dfv /o/m

/o If file2.txt exists, it is overwritten without notice.

/c Creates the new template file3.dfv.

/m The template file3.dfv exists and will be modified if necessary.

If you want the names of paragraph and character formats to be different in Word or if you want to adapt them to an existing Word template, you must execute a format conversion.

When importing RTF files into SAPscript, you should create the source text with a template and store it as RTF file. Paragraphs and characters formatted with templates can receive formats again in SAPscript. So make sure not to format texts using Word buttons! On the R/3 side, a style or form should exist whose character and paragraph formats correspond to those of the template and have the same keys. If this is not possible because the template has other keys, you can start a **format conversion**.

ASCII Files

In ASCII files, the text is stored unformatted. The only formatting element is the line feed. Character set conversions into and from all character sets defined in the spool administration are supported.

From within the SAP*script* editor, you can

- export SAP*script* texts into a local file in the formats ITF, RTF, or ASCII,
- import local files of the formats ITF, RTF, or ASCII and insert them at the cursor position as ITF text.

To find these functions in the text editor, choose *Text → Upload...* / *Text → Download...*, *Documentation → Upload...* / *Documentation → Download...* or *Clipboard→ Upload...* / *Clipboard → Download...*.

Converting SAPscript Texts

From within application programs, you can call these and further functions using a set of function modules.

Programming Example

Convert a SAP*script* standard text in system language into an RTF file and write it into a local directory. You can choose the text name and the name of the target file at will; default values are "SAPSCRIPT-DRUCKERTEST" and as local directory "C:\temp\". The system does not check whether the file name ends with ".rtf"; you must check this yourself.

```
REPORT YCMTESTE LINE-SIZE 255 MESSAGE-ID TD.
PARAMETERS:
  TEXTNAME LIKE THEAD-TDNAME    DEFAULT 'SAPSCRIPT-
DRUCKERTEST',
  FILE     LIKE RLGRAP-FILENAME DEFAULT 'C:\temp\'.
DATA: TEXTHEADER LIKE THEAD.
DATA: TEXTLINES  LIKE TLINE OCCURS 100 WITH HEADER LINE.
CALL FUNCTION  'READ_TEXT'
  EXPORTING  NAME     = TEXTNAME
             LANGUAGE = SY-LANGU
             OBJECT   = 'TEXT'
             ID       = 'ST  '
  IMPORTING  HEADER   = TEXTHEADER
  TABLES     LINES    = TEXTLINES
  EXCEPTIONS OTHERS   = 1.
CHECK SY-SUBRC = 0.
CALL FUNCTION  'EXPORT_TEXT'
  EXPORTING  CODEPAGE         = '1133'
             FILE             = FILE
             FORMATWIDTH      = 132
             FORMAT_TYPE      = 'RTF'
             HEADER           = TEXTHEADER
             SSHEET           = ' '
             WITH_TAB         = ' '
  TABLES     ITF_LINES        = TEXTLINES
  EXCEPTIONS DOWNLOAD_ERROR   = 1
             FILE_OPEN_ERROR  = 2
             FILE_WRITE_ERROR = 3.
```

```
CASE SY-SUBRC.
  WHEN 0. MESSAGE S807 WITH FILE.
  WHEN 1. MESSAGE E815 WITH FILE.
  WHEN 2. MESSAGE E811 WITH FILE.
  WHEN 3. MESSAGE E814 WITH FILE.
ENDCASE.
```

Consistency Checks

At the interface to its function modules, SAPscript implicitly always checks whether the specified text object, text name, text ID, and text language are valid. To execute such a check independent of a SAP*scrip*t processing function, the system provides function modules that allow you to explicitly check the validity of these specifications.

Printing Texts

To print a text, you can use the function module PRINT_TEXT. However, you can format only one text with this function module. Formatting occurs according to the formatting information specified in the text header (style and form). If no form is defined for a text, the system implicitly underlays the form SYSTEM.

During formatting, the system

- formats the individual text paragraphs according to the definitions in the style or form,
- replaces variables with their current values,
- interprets and executes the control statements contained in the text,
- automatically triggers page breaks according to the page layout specified in the form.

The function module automatically opens the form and closes it after formatting the text lines. All text lines transferred appear in the main window of the form (MAIN).

If you want to output a text module in such a way that you can view it in the text editor in ITF format, use the function module PRINT_TEXT_ITF. The system ignores the style and form specifications in the text header and uses the form SAPSCRIPT_ITF instead.

With each application program, you can specify different formatting parameters by using the parameter OPTIONS. The user gets a dialog window, in which he can change the default values of certain parameters. You can evaluate these changes by comparing the fields of the parameter OPTION with the corresponding fields of the return parameter RESULT of the program.

The function module PRINT_TEXT internally calls the function modules OPEN_FORM, WRITE_FORM_LINES, and CLOSE_FORM. Therefore, you cannot call PRINT_TEXT after a form has been opened using OPEN_FORM. The system then ends the function module with the exception UNCLOSED.

If the text module you want to print is not in ITF format, that is, the field TDTEXTTYPE of the text header is not empty, you cannot use the SAPscript composer to format this text. Instead, the system calls the function module PRINT_TEXT_FORMAT_xxx, which calls the word processing program appropriate for processing the specified text format. If this is impossible, PRINT_TEXT ends with an exception.

SAPscript Control Tables and Structures

This topic introduces the control tables SAPscript employs. You also find an overview of the structures used.

Control Tables

TTXOB [Page 127]	Definition of the text objects
TTXOT [Page 129]	Descriptions of the text objects
TTXID [Page 130]	Definition of the text IDs
TTXIT [Page 132]	Descriptions of the text IDs

To maintain the above tables, use transaction SE75. To call the transaction, choose *Tools* → *Word processing* → *Settings*

SAPscript Structures [Page 133]

TTXOB: Definition of the Text Objects

The control table TTXOB contains the definitions of the text objects supported by SAPscript. The table's key is the 10-digit text object. Using the SAPscript function modules, you can process only texts whose objects are stored in this table. If you pass an object name to a SAPscript function module that is not defined in table TTXOB, the function module triggers the exception OBJECT.

The table is client-independent. To maintain it, choose *Tools* → *Word processing* → *Settings* to call transaction SE75.

TDSAVEMODE: storage mode

You can enter the following values in this field:

'D'	direct storage
'V'	storage in update task
' '	text not stored in the text file

⚠

Usually, you are not allowed to change this parameter, since the application programs must execute extra functions depending on the storage mode (for example, COMMIT_TEXT).

TDAPPL: editor interface

The parameter specifies the editor interface set when a text with this object is edited in the SAPscript text editor. The interface must be defined for program SAPLSTXX.

The specified interface determines the first two letters of a status set. Depending on the processing mode (change, display, insert, mark), the SAPscript editor supplies the last two characters of the four-digit status name.

TDLINESIZE: line width of the editor

Maximum line width allowed for a text in the editor.

TDSTYLE: default style

If you create a new text module, the system automatically allocates the style specified in this field. The style determines the paragraph and character formats. Depending on the editor interface, you can change this style in retrospect.

TDFORM: default form

If you create a new text module, the system proposes the form specified in this field. It is used, if you print the text module from within the text editor. If you print the text module from within the application program, you can specify another form when calling the corresponding function module. Depending on the text interface, you may be allowed to change the default form in the text editor.

If you do not define a default form, SAPscript uses the form SYSTEM.

TDTEXTTYPE: text format

This field determines the format of the text. Depending on the format, the system calls the appropriate editor. You can specify the format both for a text object in table TTXOB and for a text ID in table TTXID. If you use table TTXOB to specify a format, it applies to all IDs that belong to this object.

If you specify a format neither in table TTXOB not in table TTXID, the system interprets the text in ITF format and calls the SAPscript editor.

TTXOT: Description of the Text Objects

This table contains a language-dependent short text used to describe the meaning of a text object.

The table is client-independent. To maintain the table, call transaction SE75 by choosing *Tools* → *Word processing* → *Settings.*

TTXID: Definition of Text IDs

This control table contains all text IDs supported by SAPscript. Text IDs are always allocated to a text object. The key of this table thus consists of the text object and a four-digit text ID. You can use the SAPscript function modules only to process texts whose IDs are stored in this table.

The table is client-independent. To maintain the table, call transaction SE75 by choosing *Tools* → *Word processing* → *Settings*.

TDINCLID: include ID

You use the INCLUDE statement to include the contents of a second text into the first text. To specify the text you want to include, you must also enter the text ID. If it is missing in the INCLUDE statement, the system uses the default ID (if any) which you can specify in this field.

TDSHOWNAME: display text name

This field determines whether the name of a text appears in the title line of the SAPscript text editor together with other information.

By default, this title line consists of the following fields:

<text ID> <activity>: *<text name> <text> Language <language key>*

<*text ID*>: long text of the text ID from table TTXIT

<*activity*>: The activity is determined by the call of the text editor or by the current activity within the text editor (change, display, insert, mark).

<*text name*>: name of the currently edited text

<*text*>: additional text, which the application program may specify when calling the text editor

<*language key*>: language key of the currently edited text

If the field is empty, the system does not display the text name in the title line of the editor. This allows you to set any title text in the editor when calling EDIT_TEXT [Page 192] with the parameter EDITOR_TITLE. The system displays this title in addition to the other headings generated by SAPscript in the title bar. You can use this to display difficult technical text names in the title bar in decrypted form.

TDTEXTTYPE: text format

This field determines the format of the text. If in table TTXOB the field TDTEXTTYPE is empty for the object to which a text ID belongs, the system uses the format specified here for the text ID. Otherwise, the entry made for the text object prevails.

If this field is empty, the system interprets the text in ITF format and calls the SAPscript editor.

TDKEYSTRUC: structure of the text key

In this field, you can store the name of a structure which describes the key structure of the name field of the text module. This field is used for documentation only and is not evaluated by SAPscript.

TDOBLIGAT: reserve

SAPscript does currently not evaluate this field.

TDDELPROT: reserve

SAPscript does currently not evaluate this field.

TDINCLRES: reserve

SAPscript does currently not evaluate this field.

TTXIT: Description of the Text IDs

This table contains a language-dependent short text used to describe the meaning of a text ID.

The table is client-independent. To maintain the table, call transaction SE75 by choosing *Tools* → *Word processing* → *Settings*.

SAPscript Structures

THEAD	Structure for storing the text header
TLINE	Structure for storing a text line
ITCPO	Control parameter for output formatting
ITCPP	Return parameter from output formatting
ITCED	Control parameter for the text editor
ITCER	Return parameter from the text editor
ITCST	Structure of the result table that gathers all the symbols appearing in a text
ITCWE	Structure of the result table that lists all the elements appearing in a form

Print Output

Controlling the print output is a most interesting topic and is treated in detail in the following.

Controlling Print Output [Page 135]

Return Parameters of the Print Output [Page 144]

Controlling Print Output

With the function modules PRINT_TEXT and OPEN_FORM, you can set output formatting and print control using the parameter OPTIONS. The data you pass to this parameter must have the structure ITCPO. The fields of this structure come from the areas SAPscript, the spool, and SAPcomm. Some of these fields can be changed by the user on the selection screen, if you requested it using the parameter DIALOG with the above function modules. The print program evaluates these changes using the corresponding fields of the parameter RESULT.

TDPAGESLCT	SAPscript: select print page
TDPREVIEW	SAPscript: show print view
TDNOPREV	SAPscript: disable print view
TDNOPRINT	SAPscript: disable printing from within print view
TDTITLE	SAPscript: text for tiltle line in the output selection screen
TDPROGRAM	SAPscript: program name for replacing symbols
TDTEST	SAPscript: test printout
TDIEXIT	SAPscript: return immediately after printing
TDGETOTF	SAPscript: return OTF table, no print output
TDSCRNPOS	SAPscript: display position of OTF on screen
TDDEST	Spool: name of the output device
TDPRINTE	Spool: name of the device type
TDCOPIES	Spool: number of copies
TDNEWID	Spool: new request
TDIMMED	Spool: print request immediately
TDDELETE	Spool: delete request after printing
TDLIFETIME	Spool: retention time of the request
TDDATASET	Spool: identification of the request
TDSUFFIX1	Spool: suffix 1 of the request
TDSUFFIX2	Spool: suffix 2 of the request
TDAUTORITY	Spool: authorization for a request
TDARMOD	Spool: archiving mode
TDCOVER	Spool: print cover page
TDCOVTITLE	Spool: cover page: title text
TDRECEIVER	Spool: cover page: recipient name

TDDIVISION	Spool: cover page: division name
TDSCHEDULE	SAPcomm: type of scheduled send time
TDSENDDATE	SAPcomm: requested send date
TDSENDTIME	SAPcomm: requested send time
TDTELELAND	SAPcomm: country key of recipient country
TDTELENUM	SAPcomm: telecommunications partner

TDPAGESLCT: SAPscript: select print page

The field contains the page specifications of the pages to be printed. You can specify individual pages, page intervals, and a combination of both. If the field is empty, the system prints all pages.

The page numbers refer to the physical pages of the SAPscript printout, not to the logical page numbering of the form.

Example:

4	only page 4
2-5	pages 2 to 5, included
20	from beginning to page 20, included
3-	from page 3 to end

To combine these variants, separate the different specifications with commas:

- 4,8-10,15-

  prints the pages 4, 8 to 10, and then from page 15 to the end.

The user can change the values proposed in this field on the selection screen.

TDPREVIEW: SAPscript: print view

The field determines whether SAPscript shall create a print view. You can then see on the screen exactly what the printout will look like later. In background processing, the system does not interpret this field. It always creates a spool request.

Possible values:

‘X’	print view wanted
‘ ‘	no print view wanted

TDNOPREV: SAPscript: disable print view

On the print selection screen, the user can choose to display the print view of the SAPscript output on the screen. If you want to disable this function, use field TDNOPREV.

Possible values:

'X'	disable print view function
' '	allow print view function

TDNOPRINT: SAPscript: disable printing from within print view

Use this field to determine whether the user is allowed to start the printing process from within the print view of a SAPscript text. If, for example, the application program shall control print output independent of any user entries, activate this field.

Possible values:

'X'	printing from within print view disabled
' '	Printing from within print view allowed

TDTITLE: SAPscript: text for title line of print selection screen

The text in this field is displayed in the title line of the print selection screen.

TDPROGRAM: SAPscript: program name for replacing symbols

To replace program symbols, SAPscript must know in which active program to find the work areas for the corresponding values. If no program is specified in this field, the system looks in the program that was called first (program name from SY-CPROG); otherwise in the data area of the program specified here.

The program name is valid during the entire print process. With form printing, however, you can specify a new program name when calling the function module START_FORM. This new program name is then valid until the next END_FORM. Afterwards, the system uses the program name specified in TDPROGRAM again.

SAPscript accesses the table fields of this program using a dynamic ASSIGN with the program name specified here. If no program name is entered or if the program has not been loaded yet, the system treats the symbol as text symbol.

TDTEST: SAPscript: test printout

Use this field to format a text for printing in a test mode. This means that the system does not replace the symbols in the text with their current values. Instead, it represents all output positions of a symbol value using 'X'.

Possible values:

'X'	format text in test mode
' '	format text as usual

TDIEXIT: SAPscript: return immediately after printing

Use this parameter to determine whether to return to the application program immediately after printing the text from within the print view. Usually, the system remains in the print view.

Returning to the application program may be necessary, if a text is to be printed only once or if the print view is no longer needed after printing.

Possible values:

‘X’	leave print view immediately after printing
‘ ‘	remain in print view after printing

TDGETOTF: SAPscript: return the OTF table

If you enter ‘X’ in this field, the SAPscript composer produces the print format as usual, but does not pass it to the spool or print view. Instead, it passes the OTF format created to the calling program for further processing, using the table parameter OTFDATA of the function modules PRINT_TEXT and CLOSE_FORM.

Possible values:

‘X’	return output format
‘ ‘	pass output format to spool or print view

TDDEST: Spool: name of the output device

Specify the name of the device on which you want to output the formatted SAPscript text. If you enter ‘*’ or leave the field empty, the system uses the default value specific in the user master record of the active user. If, in this case, no printer is specified in the user master record, the system displays the print selection screen, even if you wanted to suppress the dialog when calling the function modules PRINT_TEXT or OPEN_FORM (parameter DIALOG).

The user can change the value proposed on the selection screen.

TDPRINTER: Spool: name of the device type

Usually, this field is empty. On the selection screen, the user can choose among all existing printers. However, if you want to ensure that the text is output on printers of a certain type only, you can specify the device type in this field. On the selection screen, the system then offers only the printers of this type.

The device type you specify must be defined in table TSP0A. To find these types out, use the spool administration (transaction SPAD).

TDCOPIES: Spool: number of copies

Specify how often you want the spool to print a text. ‘1’ means that the entire text is printed once (default). If you specify '2', the system prints all pages twice. SAPscript internally replaces the value '0' with ‘1’.

When you create more than one copy, the sequence of the printed pages is 1-2-3..., 1-2-3...

The user can change the proposed value on the print selection screen.

TDNEWID: Spool: new spool request

This field determines whether to append the current spool request to an existing request with the same attributes or whether to create a new request. To append a request to another, the values of the fields *Name, Output device, Number of copies* and the *Formatting mode* must be the same, and the existing spool request must still be active. This is no longer the case, if a spool request is released to printing. If the system does not find a matching spool request, it always creates a new one.

Possible values:

‘X’	create a new spool request
‘ ‘	find a matching spool request for appending

The user can change the proposed value on the print selection screen.

TDIMMED: Spool: print request immediately

Use this field to determine whether to send the print request to the output device immediately after completing it. Otherwise, you must use the spool print control (transaction SP01) to release the print request.

Possible values:

‘X’	print request immediately after completing it
‘ ‘	keep request in spool after completing it

The user can change the proposed value on the print selection screen.

TDDELETE: Spool: delete request after printing

Use this field to determine whether to delete the spool request immediately after printing it on the output device or whether to keep it for the spool retention period.

Possible values:

‘X’	delete immediately after printing
‘ ‘	delete after retention time has expired

The user can change the proposed value on the print selection screen.

TDLIFETIME: Spool. retention time of the request

This field determines for how many days the system keeps a request in the spool before deleting it. If the field is empty, SAPscript inserts the default value '8'.

The user can change the proposed value on the print selection screen.

TDDATASET: Spool: identification of the request

The field is the first component of the three-part identification of the spool request (including also the fields TDSUFFIX1 and TDSUFFIX2). There is no naming convention for the identification. If your application uses a certain convention, see the corresponding application documentation.

If the field is empty, SAPscript enters the value SCRIPT.

The user can change the proposed value on the print selection screen.

TDSUFFIX1: Spool: suffix 1 of the request

Second part of the identification of the spool request. See also the description of field TDDATASET.

If the field is empty, SAPscript enters the output destination (TDDEST).

The user can change the proposed value on the print selection screen.

TDSUFFIX2: Spool: suffix 2 of the request

Third part of the identification of the spool request. See also the description of field TDDATASET.

If the field is empty, SAPscript enters the name of the user.

The user can change the proposed value on the print selection screen.

TDAUTORITY: Spool: authorization for request

This field defines an authorization value for the spool request. Only users with the specified authorization can display or print the contents of the spool request.

The spool print control (transaction SP01) checks whether the authorization object S_SPO_ACT (spool actions) of the user contains the specified value.

The user can change the proposed value on the print selection screen.

TDARMOD: Spool: archiving mode

Use this field to determine whether only to print a request or whether to store it in the optical archive as well.

Possible values:

‘1’	only print request (default)
‘2’	only archive request

‘3’	print and archive request

The user can change the proposed value on the print selection screen.

TDCOVER: Spool: print cover page

This field determines whether the printout includes a cover page containing information such as recipient name, division name, format used, and so on.

Possible values:

‘X’	print cover page
‘ ‘	suppress cover page
‘D’	print cover page depending on the setting of the respective output device. (see definition of the device in the spool administration (transaction SPAD), in the column *output devices*)

The user can change the proposed value on the print selection screen.

TDCOVTITLE: Spool: cover page: title text

This field contains a text describing the spool request. It appears on the cover page.

The user can change the proposed value on the print selection screen.

TDRECEIVER: Spool: cover page: recipient name

You can specify the name of the user who receives the spool request. The system prints this name on the cover page. The default value is the name of the current user.

The user can change the proposed value on the print selection screen.

TDDIVISION: Spool: cover page: division name

This field contains the name of the division to which the user belongs. The system prints this name on the cover page.

The user can change the proposed value on the print selection screen.

TDSCHEDULE: SAPcomm: type of scheduled send time

Use this field to determine whether to send a spool request via the SAP communication interface immediately or whether to wait for the night.

Possible values:

‘IMM’	send request immediately
‘NIG‘	send request during the night

If the field is empty, the system uses the default value ‘IMM’.

The user can change the proposed value on the print selection screen.

TDSENDDATE: SAPcomm: requested send date

In this field, enter the date on which to send the print request via the SAP communication interface.

The user can change the proposed value on the print selection screen.

TDSENDTIME: SAPcomm: requested send time

In this field, enter the time at which to send the print request via the SAP communication interface.

The user can change the proposed value on the print selection screen.

TDTELELAND: SAPcomm: country key for recipient country

According to the country key specified in this field, the SAP communication interface determines the country-specific area code and uses it as prefix to the telephone number of the telecommunications partner specified in field TDTELENUM.

The user can change the proposed value on the print selection screen.

TDTELENUM: SAPcomm: number of telecommunications partner

In this field, enter the number of the desired telecommunications partner is the way it is dialed in the recipient country. The system automatically includes the area code which you specify in the field TDTELELAND.

To switch off automatic number check and prefixing with the area code, start the number in this field with '&'. In this case, you must specify the entire number, including area code, but without operator call.

The telephone number must be of a certain format:

TELEFAX

Allows only digits and the characters ‘(‘, ‘)’, ‘/’, ‘-’, and ‘.' as well as blanks.

TELEX

Allows only digits, the letters A to Z, and blanks. The number must have the following structure:

nnn...n aaaa..a ccc

'n'	digits that form the numeric part of the telex identification
‘a'	letters that for the alphanumeric part of the telex identification
‘ccc’	consists of one, two, or three letters and corresponds to the country key

TELETEX

Allows only digits and the letter A to Z. The number must have the following structure:

nnn...n=aaaa..a

‘n'	digits thst form the numeric part of the teletex identification
'a'	etters that form the alphanumeric part of the teletex identification

The user can change the proposed value on the print selection screen.

Return Parameters of the Print Output

After formatting SAPscript texts for printing, the parameter RESULT contains information and settings which can be of interest for the calling program. This information can be passed using the function modules PRINT_TEXT or CLOSE_FORM. The parameter reference structure is ITCPP.

The structure includes fields, which have previously been passed in the parameter OPTIONS when calling the output function module, and other information, which the system can supply only after completing the formatting request.

Differences between the values in the parameter OPTIONS and the corresponding fields in the parameter RESULT tell the application program that the user changed the default values displayed in the print selection screen.

TDPAGESLCT	SAPscript: select print page
TDNOPREV	SAPscript: disable print view
TDPREVIEW	SAPscript: print view
TDNOPRINT	SAPscript: disable print function from within print view
TDTITLE	SAPscript: text for title line in the print selection screen
TDPROGRAM	SAPscript: program name for replacing symbols
TDTEST	SAPscript: test printout
TDIEXIT	SAPscript: return immediately after printing
TDGETOTF	SAPscript: return OTF table; no print output
TDSCRNPOS	SAPscript: display position for OTF on the screen
TDAPPL	SAPscript: interface of the print view
TDOTFCALL	SAPscript: name of the driver module
TDOTFTYPE	SAPscript: OTF type
TDPAGES	SAPscript: number of printed pages
TDFORMS	SAPscript: number of used forms
TDWARNINGS	SAPscript: number of warnings during print formatting
TDDEVICE	SAPscript: type of output device
TDSCREEN	SAPscript: type of screen display
TDSCDRIVER	SAPscript: type of screen driver
TDSCABAP	SAPscript: ABAP list as print view
USEREXIT	SAPscript: last executed user function
TDRTL	SAPscript: right-to-left language in OTF

TDDEST	Spool: name of the output device
TDPRINTER	Spool: name of device type
TDCOPIES	Spool: number of copies
TDNEWID	Spool: new request
TDIMMED	Spool: print request immediately
TDDELETE	Spool: delete request after printing
TDLIFETIME	Spool: retention time of the request
TDDATASET	Spool: identification of the request
TDSUFFIX1	Spool: suffix 1 of the request
TDSUFFIX2	Spool: suffix 2 of the request
TDAUTORITY	Spool: authorization for a request
TDARMOD	Spool: archiving mode
TDCOVER	Spool: print cover page
TDCOVTITLE	Spool: cover page: title text
TDRECEIVER	Spool: cover page: recipient name
TDDIVISION	Spool: cover page: division name
TDSPOOLID	Spool: number of the request
TDDRIVER	Spool: name of a driver
TDABAP	Spool: driver type
TDPAGEFORM	Spool: page format of the spool request
TDSCHEDULE	SAPcomm: type of scheduled send time
TDSENDDATE	SAPcomm: requested send date
TDSENDTIME	SAPcomm: requested send time
TDTELELAND	SAPcomm: country key of recipient country
TDTELENUM	SAPcomm: number of telecommunications partner
TDTELENUME	SAPcomm: dialed number of telecommunications partner

TDPAGESLCT: SAPscript: select print pages

The field contains the pages to be printed. The value either comes from user entries on the print selection screen or from the corresponding field passed in the OPTIONS parameter.

TDNOPREV: SAPscript: print view

Describes whether the user was able to call the print view from the print selection screen.

Possible values:

'X'	print view was disabled
' '	print view was enabled

TDPREVIEW: SAPscript: display print view

The print program uses this field to determine whether the user actually called the print view from the print selection screen.

Possible values:

'X'	user called print view
' '	print view was not called

TDNOPRINT: SAPscript: disable print function from print view

Describes whether the user was able to print the SAPscript text from within the print view.

Possible values:

'X'	print function enabled on print view display
' '	print function disabled on print view display

TDTITLE: SAPscript: title on print selection screen

The field contains the text, which was displayed in the title line of the print selection screen.

TDPROGRAM: SAPscript: program name for replacing symbols

The field returns the name of the program that was specified in the corresponding field of the OPTIONS parameter when calling the print function. This program is the default value for work areas from which the system replaced the program symbols with the current values.

TDTEST: SAPscript: test printout

Indicates whether the user used the print function in test mode.

Possible values:

'X'	formatting in test mode
' '	normal print formatting

TDIEXIT: SAPscript: return after printing

The field indicates whether after printing from within the print view, the system immediately returned to the application program.

Possible values:

'X'	print view left immediately after printing
' '	remained in print view after printing

TDGETOTF: SAPscript: return of OTF table; no print output

The field indicates whether the SAPscript formatting program passed the print output (OTF) to the print view or spool or whether it returned the output to the calling program using table OTFDATA.

Possible values:

'X'	print output returned in table OTFDATA
' '	print output passed to print view or spool

TDSCRNPOS: SAPscript: display position for OTF on the screen

The field returns the display position on the screen, which was specified in the call in the corresponding field of the OPTIONS parameter.

TDAPPL: SAPscript: interface of the print view

The field returns the abbreviation of the interface name used for the print view. The abbreviation corresponds to the editor interface name you can specify for a word processing application object in table TTXOB. The contents corresponds to the value passed in the parameter APPLICATION when calling the function modules OPEN_FORM or PRINT_TEXT.

TDOTFCALL: SAPscript: name of driver function module

The field contains the name of the ABAP function module used as driver to convert the OTF format into the device-specific control sequences. This field is filled only if in table TSP09 an entry for the current output device specifies that the system shall use an ABAP function module as driver.

⇨

Usually, this field is empty, since in recent releases the output drivers are C functions.

TDOTFTYPE: SAPscript: driver type for OTF output

This field specifies the type of the OTF driver used to print a SAPscript document. The format SAPscript creates after formatting a text for outputting is called OTF format (Output Text Format). This format is independent of the output device and must therefore be converted before outputting the text on a certain device (screen, printer type). During this conversion, the system, that is, the SAPscript drivers, replace the OTF commands with device-specific control sequences. The names of the drivers are defined in table TSP09. Table TSP0A is used to allocate a driver to each device type (including printers, screens, telefaxes, and so on). This driver then formats SAPscript texts for this device.

At present, the following OTF drivers exist:

- STN2 for normal line printers
- PRES for Kyocera PRESCRIBE printers
- POST for Postscript printers
- HPL2 for HP LaserJet II printers

TDPAGES: SAPscript: number of printed pages

This field specifies the number of physical pages created for screen display or print output. This number includes all copies, even though they are created only by the spool system and cannot be seen in the print view. A cover page is not included into the number stored in TDPAGES.

TDFORMS: SAPscript: number of used forms

The contents of this field indicates how many forms were started during SAPscript formatting. If you called the function module PRINT_TEXT for printing, this field always contains 1. If you used explicit form control, the system counts all forms called using the function module START_FORM within a OPEN_FORM / CLOSE_FORM chain.

TDWARNINGS: SAPscript: warnings during formatting

This field contains the number of warnings that occurred during print formatting. Warnings in SAPscript do not end the print formatting process. Depending on the type of warning, the system either ignores the cause or uses default values.

Possible warnings are:

- Character format not defined
- Paragraph format not defined
- The system encountered the end sequence </> without a corresponding character format.
- In the current paragraph, the end sequence </> of a character format is missing.
- The system could not find a text the user wanted to INCLUDE.
- A specified character is not defined in the current print font.
- An invalid text statement has been called.

TDDEVICE: SAPscript: type of output device

The field returns the type of the output device used. Its contents corresponds to the value of the parameters DEVICE, which you can specify when calling the function modules PRINT_TEXT and OPEN_FORM

Possible values:

'PRINTER'	Formatting for the specified printer
'TELEX'	Formatting for telex output
'TELEFAX'	Formatting for telefax output
'SCREEN'	Formatting for screen output as ABAP list. The parameter APPLICATION determines the interface.
'ABAP'	Formatting for screen output as ABAP list. The calling program controls the interface.
'OTF_MEM'	The OTF format created by SAPscript is stored in the text memory. Formatting as for SCREEN

TDSCREEN: SAPscript: type of screen display

Returns the type of screen display. The value ‘SCREEN’ is returned only if in DEVICE, formatting for screen output is specified. All other values refer to the value ‘PRINTER’ of the field DEVICE and appear only if the user selected the print view as screen display.

Possible values:

' '	no screen display
'SCREEN'	formatting for screen as ABAP list
'MF'	print view display for Motif
'PM'	print view display for OS/2
'WN'	print view display for Windows
'WN32'	print view display for Windows NT

TDSCDRIVER: SAPscript: type of screen driver

Returns the type of the driver used to create the screen display of the print view.

Possible values:

' '	no screen display
'LIST'	ABAP list
'MF'	print view for Motif
'PM'	print view for OS/2
'WN'	print view for Windows

'WN32'	Print view for Windows NT

TDSCABAP: SAPscript: ABAP list as print view

The field indicates whether the SAPscript print view was represented as ABAP list.

Possible values:

'X'	ABAP list
' '	no ABAP list

USEREXIT: SAPscript: last executed user action

The field indicates the function the user chose to leave the print selection screen.

Possible values:

'C'	user chose function *Cancel*
'B'	user chose function *Back*
'E'	user chose function *Exit*

TDRTL: SAPscript: right-to-left language in OTF

The field indicates whether the print output contains text lines in a language whose characters are output from right to left (fro example, Hebrew).

Possible values:

'X'	right-to-left language
' '	no right-to-left language

TDDEST: Spool: name of output device

The field contains the name of the output device for which the system formatted the output.

TDPRINTER: Spool: name of device type

The field contains the type of the device specified in TDDEST.

TDCOPIES: Spool: number of copies

The field indicates how often the output was printed.

The user can set the contents of this field on the print selection screen.

TDNEWID: Spool: new request

Tells the print program whether to append the output to an existing request or whether to create a new request anyway.

The user can set this option on the print selection screen.

Possible values:

'X'	new spool request
' '	append to existing spool request

TDIMMED: Spool: print request immediately

The field indicates whether the request was printed immediately after completion or whether it was kept in the spool until the user or the program explicitly triggered print output.

The user can set this option on the print selection screen.

Possible values:

'X'	request printed immediately
' '	request kept in spool

TDDELETE: Spool: delete request after printing

Indicates whether the spool request shall be deleted after printing.

The user can set this option on the print selection screen.

Possible values:

'X'	delete request after printing
' '	keep request

TDLIFETIME: Spool: retention period of the request

The value corresponds to the number of days for which the request is kept in the spool.

The user can set this option on the print selection screen.

TDDATASET: Spool: name of the request

Returns the first part of the three-part identification of the spool request.

The user can set this option on the print selection screen.

TDSUFFIX1: Spool: suffix 1 of the request

Returns the second part of the three-part identification of the spool request.

The user can set this option on the print selection screen.

TDSUFFIX2: Spool: suffix 2 of the request

Returns the third part of the three-part identification of the request.

The user can set this option on the print selection screen.

TDARMOD: Spool: archiving mode

The print program uses this field to determine the archiving mode the user set on the print selection screen.

Possible values:

'1'	request printed (default)
'2'	request archived
'3'	request printed and archived

TDCOVER: Spool: print cover page

Indicates whether a cover page was printed.

The user can set this option on the print selection screen.

Possible values:

'X'	cover page printed
' '	no cover page printed
'D'	cover page printed according to the default setting of the output device (see definition of the device in the spool administration, transaction SPAD, column *output device*).

TDCOVTITLE: Spool: cover page: title text

Contains the title text for the cover page

The user can enter the title text on the print selection screen.

TDRECEIVER: Spool: cover page: recipient name

Contains the recipient name for the cover page of the print request, which the user entered on the print selection screen.

TDDIVISION: Spool: cover page: division name

Contains the division name for the cover page of the print request, which the user entered on the print selection screen.

TDAUTORITY: Spool: print authorization

The field specifies the authorization a user must have to display the print request in the spool.

The user can set this option on the print selection screen.

The spool print control (transaction SP01) checks whether a user has this value stored in the authorization object S_SPO_ACT (spool actions).

TDSPOOLID: Spool: number of the request

The field returns the number of the spool request into which the system placed the print output. The spool system assigns this number. If the value is > 0, the print program knows, that the system actually printed the request.

TDDRIVER: Spool: name of the driver

Contains the name of the driver that converted the SAPscript output format OTF into the final printer control sequences.

TDABAP: Spool: driver type

Returns the type of the SAPscript printer driver used for output.

Possible values:

'X'	printer driver in ABAP
' '	printer driver in C

TDPAGEFORM: Spool: page format of the request

The field describes the page format of the print request. The page format is determined by the forms called in the current print request. One print request can contain only output with the same page format.

The possible page formats are described in table TSP08, which you can display using the spool administration (transaction SPAD, page formats).

TDSCHEDULE: SAPcomm: type of scheduled send time

Returns the type of send time of a print request sent via the SAP communication interface.

Possible values:

'IMM'	send request immediately
'NIG'	send request during the night

TDSENDDATE: SAPcomm: requested send date

The field contains the date on which the request shall be sent via the SAP communication interface.

TDSENDTIME: SAPcomm: requested send time

The field contains the time at which the request shall be sent via the SAP communication interface.

TDTELELAND: SAPcomm: country key

Specifies the country key of the desired telecommunications partner. According to this key, the system includes other country-specific information into the telephone number of the telecommunications partner.

TDTELENUM: SAPcomm: number of the telecommunications partner

Contains the originally specified number of the telecommunications partner.

TDTELENUME: SAPcomm: dialed number of the telecommunications partner

This field contains the complete telephone number of the telecommunications partner in the form in which the SAP communication interface constructed it according to the specified country key.

Editor Control

The following topics explain how you can control the editor and which parameters the editor uses to return values and messages.

Controlling the Editor [Page 156]

Return Parameter of the Editor [Page 159]

Controlling the Editor

When calling the SAPscript text editor using the function modules EDIT_TEXT or EDIT_TEXT_INLINE, you can use the parameter CONTROL to set certain attributes of the editor. Some of these attributes depend on the editor interface used. The parameter CONTROL uses the structure ITCED.

NOENDLINES: no blank lines at the end of the text

Using this field, you can determine whether you want the system to automatically insert blank lines at the end of a text up to the bottom of the screen. These blank lines are ready to accept input.

These blank lines facilitate entering text at the end of the original text. The system automatically deletes any unused lines when saving the text.

Possible values:

‘X’	insert no blank line
‘ ‘	fill screen window with input-enabled blank lines (default)

SCROLLEND: position cursor at text end

Use this field to indicate whether you want the system to automatically position the cursor at the end of the text whenever you call the SAPscript editor. By default, the system displays the beginning of the text, starting with the first line of text, and positions the cursor on the first column of the first line.

Possible values:

'X'	position cursor at end of text
' '	display text from the first line (default)

USERTITLE: suppress SAPscript status information

SAPscript display the following status information in the tile line:

- function (display or change)
- text description from table TTXIT
- text name, if required by table TTXID

The calling program can pass additional information in the parameter EDITOR_TITLE when calling the function modules EDIT_TEXT or EDIT_TEXT_INLINE. The system displays this title text in addition to the SAPscript status information (default).

Use the parameter USERTITLE to suppress the SAPscript status information.

Possible values:

'X'	suppress SAPscript status information

' '	include SAPscript status information into the title (default)

If you suppress SAPscript status information, the system replaces a & character in the parameter EDITOR_TITLE with the editing function (display, change).

SHOWTPFM: display the format of template lines

Use the paragraph format '>' to define the contents of a line as template line. The system then highlights the line contents and disables the field for input. This function allows you to separate the text into different sections whose separator lines the user cannot change.

Usually, you want these template lines to appear in the printout as well. Therefore, you must place the paragraph format of the template line into the first two columns of the line. The editor does not display this paragraph format.

If you want to see the paragraph format in the editor, request it using the parameter SHOWTPFM.

Possible values:

'X'	display format of template lines
' '	suppress format information of template lines

APP_NEXT: activate menu function *Next text*

The calling program tells the SAPscript editor that a subsequent text exists for the current text. This activates the menu function *Goto* → *Next text* in the editor.

Possible values:

'X'	subsequent text exists
' '	no subsequent text exists

⚠

This field applies only to texts edited on the TA interface.

APP_PREV: activate menu function *Previous text*

The calling program tells the SAPscript editor that a previous text exists for the current text. This activates the menu function *Goto* → *Previous text* in the editor.

Possible values:

'X'	Previous text exists
' '	no previous text exists

⚠

This field applies only to texts edited on the TA interface.

APP_SUBID: use existing editor interfaces for own applications

The editor interface is set according to the interface assigned to the text object. If you want to change an existing interface, you can specify an application SUBID with reference type TDAPP.

⚠

When you specify the application SUBID, the system takes you to a function module of the application. Handling the respective function codes and activating or deactivating menu functions must be done by the application.

CHANGEMODE: allow switching between Create/Change

You can determine whether to allow switching between display and editing mode in the editor. If you select the parameter, the system displays the appropriate pushbutton in the editort.

⚠

You can set this parameter in the PC editor only. And, the parameter is effective for the TX interface for standard texts only.

Possible values:

'X'	allow switching between display and editing mode
' '	do not allow switching (default)

Return Parameter of the Editor

After leaving the SAPscript editor, you can use the parameter RESULT to evaluate status information. The editor returns this information in the structure ITCER.

FUNCTION: change information on the text module

This field indicates the function executed on the transferred text module.

Possible values:

'D'	The transferred text was deleted.
'I'	The system transferred an empty lines table into which text lines were inserted.
'U'	An existing text was changed and saved.
' '	The text remained unchanged.

USEREXIT: exit status of the editor

he field contains the function which the user chose to leave the SAPscript editor.

Possible values:

'C'	user chose function *Cancel*
'B'	user chose function *Back*
'E'	user chose function *Exit*
'N'	user chose function *Next text*
'P'	user chose function *Previous text*

SAPscript Function Modules

This section contains a detailed description of the function modules used by SAPscript.

Database	
READ_TEXT [Page 163]	Reads a text module and passes it to the specified work areas.
READ_TEXT_INLINE [Page 166]	Like READ_TEXT. In addition, it passes the first few text lines to a second lines table.
READ_REFERENCE_LINES [Page 169]	Reads the text lines of a reference text and passes them to the specified lines table.
SAVE_TEXT [Page 171]	Saves a text.
DELETE_TEXT [Page 174]	Deletes a text.
COPY_TEXTS [Page 176]	Copies a text.
SELECT_TEXT [Page 179]	Finds the texts for an application object.
Administration	
REFER_TEXT [Page 182]	Creates a reference to another text.
RENAME_TEXT [Page 185]	Renames the text in the text memory.
COMMIT_TEXT [Page 187]	Creates for all texts in the text memory the call of an appropriate update module.
INIT_TEXT [Page 190]	Initializes the internal work areas for a text.
Editor call	
EDIT_TEXT [Page 192]	Calls the text editor.
EDIT_TEXT_INLINE [Page 196]	Merges the inline lines with the other text lines and calls the text editor.
Consistency check	
CHECK_TEXT_AUTHORITY [Page 200]	Checks the authorization for standard texts.
CHECK_TEXT_ID [Page 202]	Checks whether the specified text ID is valid.
CHECK_TEXT_LANGUAGE [Page 203]	Checks whether the specified text language is valid.
CHECK_TEXT_OBJECT [Page 204]	Checks whether the specified text object is valid.

CHECK_TEXT_NAME [Page 205]	Checks whether the specified text name is valid.
Editing functions	
TEXT_SYMBOL_COLLECT [Page 206]	Finds the variable symbols that occur in a text.
TEXT_SYMBOL_PARSE [Page 207]	Checks whether a character string is a SAPscript symbol.
TEXT_SYMBOL_REPLACE [Page 213]	Replaces symbols in a text with their values.
TEXT_SYMBOL_SETVALUE [Page 217]	Defines the value of a text symbol.
TEXT_CONTROL_REPLACE [Page 219]	Replaces control statements in a text (IF, CASE...).
TEXT_INCLUDE_REPLACE [Page 221]	Replaces INCLUDE control statements by the text lines of the corresponding text.
Print	
PRINT_TEXT [Page 224]	Formats a text for output.
PRINT_TEXT_ITF [Page 230]	Prints a text in the internal ITF format.
Form functions	
OPEN_FORM [Page 232]	Opens the form output.
CLOSE_FORM [Page 237]	Ends the form output.
START_FORM [Page 239]	Starts a new form.
WRITE_FORM [Page 242]	Calls a form element.
WRITE_FORM_LINES [Page 246]	Writes text lines into a form.
END_FORM [Page 250]	Ends the current form.
CONTROL_FORM [Page 251]	Sends a control statement to the form.
READ_FORM_ELEMENTS [Page 252]	Finds the elements of a form.
READ_FORM_LINES [Page 254]	Passes the lines of a form elements into an internal lines table.
Conversion	
CONVERT_TEXT [Page 256]	Converts texts between different formats.

CONVERT_TEXT_R2 [Page 273]	Converts texts between R/3 format (SAPscript) and R/2 format.
CONVERT_OTF_MEMORY [Page 282]	Converts the formatted text (OTF format).
EXCHANGE_ITF [Page 261]	Exchanges the paragraph and character formats of a text with those of another style or form.
Transfer	
IMPORT_TEXT [Page 263]	Imports texts.
EXPORT_TEXT [Page 267]	Exports texts.
TRANSFER_TEXT [Page 271]	Uploads/Downloads texts.

READ_TEXT

READ_TEXT provides a text for the application program in the specified work areas.

The function module reads the desired text from the text file, the text memory, or the archive. You must fully specify the text using OBJECT, NAME, ID, and LANGUAGE. An internal work area can hold only one text; therefore, generic specifications are not allowed with these options.

After successful reading, the system places header information and text lines into the work areas specified with HEADER and LINES.

If a reference text is used, SAPscript automatically processes the reference chain and provides the text lines found in the text at the end of the chain. If an error occurs, the system leaves the function module and triggers the exception REFERENCE_CHECK.

Function call:

```
CALL FUNCTION 'READ_TEXT'
     EXPORTING  CLIENT                  = SY-MANDT
                OBJECT                  = ?...
                NAME                    = ?...
                ID                      = ?...
                LANGUAGE                = ?...
                ARCHIVE_HANDLE          = 0
     IMPORTING  HEADER                  =
     TABLES     LINES                   = ?...
     EXCEPTIONS ID                      =
                LANGUAGE                =
                NAME                    =
                NOT_FOUND               =
                OBJECT                  =
                REFERENCE_CHECK         =
                WRONG_ACCESS_TO_ARCHIVE =
```

Export parameters:

READ_TEXT

CLIENT	Specify the client under which the text is stored. If you omit this parameter, the system uses the current client as default. Reference field: SY-MANDT Default value: SY-MANDT
OBJECT	Enter the name of the text object to which the text is allocated. Table TTXOB contains the valid objects. Reference field: THEAD-TDOBJECT
NAME	Enter the name of the text module. The name may be up to 70 characters long. Its internal structure depends on the text object used. Reference field: THEAD-TDNAME
ID	Enter the text ID of the text module. Table TTXID contains the valid text IDs, depending on the text object. Reference field: THEAD-TDID
LANGUAGE	Enter the language key of the text module. The system accepts only languages that are defined in table T002. Reference field: THEAD-TDSPRAS
ARCHIVE_HANDLE	If you want to read the text from the archive, you must enter a handle here. The system uses it to access the archive. You can create the handle using the function module A**CHIVE_OPEN_FOR_READ**. The value '0' indicates that you do not want to read the text from the archive. Reference field: SY-TABIX Default value: 0

Import parameters:

HEADER	If the system finds the desired text, it returns the text header in this parameter. Structure: THEAD

Table parameters:

LINES	The table contains all text lines that belong to the text read. Structure: TLINE

Exceptions:

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The parameter LANGUAGE contains a language key that does not exist in table T002.
NAME	The parameter NAME contains the name of a text module that does not correspond to the SAPscript conventions. Possible errors: • The field contains only blanks. • The field contains the invalid characters ‘*’ or ‘,’.
OBJECT	The parameter OBJECT contains the name of a text object that does not exist in table TTXOB.
NOT_FOUND	The system did not find the specified text module.
REFERENCE_CHECK	The text module to be read has no text lines of its own but refers to the lines of another text module. This reference chain can include several levels. For the current text, the chain is interrupted, that is, one of the text modules referred to in the chain no longer exists.
WRONG_ACCESS_TO_ARCHIVE	The exception WRONG_ACCESS_TO_ARCHIVE is triggered if an archive is accessed using an incorrect or non-existing archive handle or an incorrect mode (that is, read if the archive is open for writing or vice versa).

READ_TEXT_INLINE

READ_TEXT_INLINE provides a text for the application program in the specified work areas. You must specify these work areas with all SAPscript function modules that process the text.

The function module reads a text from the text file or the text memory. You must fully specify the text using OBJECT, NAME, ID, and LANGUAGE. An internal work area can hold only one text; therefore, generic specifications are not allowed with these options.

In addition, the system transfers as many lines from the LINES table to the INLINES table as specified in the parameter INLINE_COUNT. You can use this lines table to display the first text lines on any screen for the user to modify them. At the event PAI, you must then call the function module EDIT_TEXT_INLINE.

Function call:

```
CALL FUNCTION   'READ_TEXT_INLINE'
     EXPORTING   OBJECT          = ?...
                 NAME            = ?...
                 ID              = ?...
                 LANGUAGE        = ?...
                 INLINE_COUNT    = ?...
     IMPORTING   HEADER          =
     TABLES      LINES           = ?...
                 INLINES         = ?...
     EXCEPTIONS  ID              =
                 LANGUAGE        =
                 NAME            =
                 NOT_FOUND       =
                 OBJECT          =
                 REFERENCE_CHECK =
```

Export parameters:

OBJECT	Enter the name of the text object to which the text is allocated. Table TTXOB contains the valid text objects. Reference field: THEAD-TDOBJECT

NAME	Enter the name of the text module. The name may be up to 70 characters long. Its internal structure depends on the text object used. Reference field: THEAD-TDNAME
ID	Enter the text ID of the text module. Table TTXID contains the valid text IDs, depending on the text object. Reference field: THEAD-TDID
LANGUAGE	Enter the language key of the text module. The system accepts only languages that are defined in table T002. Reference field: THEAD-TDSPRAS
INLINE_COUNT	Specify the number of lines you want the system to transfer to table INLINES from the beginning of the text lines table LINES.

Import parameters:

HEADER	If the system finds the text, it returns the table header in this parameter. Structure: THEAD

Table parameters:

LINES	The table contains all text lines that belong to the text read. Structure: TLINE
INLINES	The table contains as many lines of table LINES as specified in the parameter INLINE_COUNT. Structure: TLINE

Exceptions:

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The parameter LANGUAGE contains a language key that does not exist in table T002.

READ_TEXT_INLINE

NAME	The parameter NAME contains the name of a text module that does not correspond to the SAPscript conventions. Possible errors: • The field contains only blanks. • The field contains the invalid characters ‘*’ or ‘,’.
OBJECT	The parameter OBJECT contains the name of a text object that does not exist in table TTXOB.
NOT_FOUND	The system did not find the specified text module.
REFERENCE_CHECK	The text module to be read has no text lines of its own but refers to the lines of another text module. This reference chain can include several levels. For the current text, the chain is interrupted, that is, one of the text modules referred to in the chain no longer exists.

READ_REFERENCE_LINES

If you did not include the text lines of the reference texts into the REFER_TEXT function module, you can use READ_REFERERNCE_LINES to read them at a later time.

Function call:

```
CALL FUNCTION  'READ_REFERENCE_LINES'
     EXPORTING  HEADER          = ?...
     IMPORTING  NEWHEADER       =
     TABLES     LINES           = ?...
     EXCEPTIONS ID              =
                LANGUAGE        =
                NAME            =
                NOT_FOUND       =
                NO_REFERENCE    =
                OBJECT          =
                REFERENCE_CHECK =
```

Export parameters:

HEADER	The field contains the text header of the text that contains the reference to another text. The system reads the lines of the text specified in the fields TDREFOBJ, TDREFNAME, and TDREFID and stores them in table LINES. Structure: THEAD

Import parameters:

NEWHEADER	Contains the modified text header. The structure specified here is usually the same as the structure passed with the parameter HEADER. Structure: THEAD

Table parameters:

LINES	LINES contains all text lines of the reference text. Structure: TLINE

Exceptions:

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The parameter LANGUAGE contains a language key that does not exist in table T002.
NAME	The parameter NAME contains the name of a text module that does not correspond to the SAPscript conventions. Possible errors: • The field contains only blanks. • The field contains the invalid characters '*' or ','.
OBJECT	The parameter OBJECT contains the name of a text object that does not exist in table TTXOB.
NOT_FOUND	The system did not find the specified text module.
REFERENCE_CHECK	The text module to be read has no text lines of its own but refers to the lines of another text module. This reference chain can include several levels. For the current text, the chain is interrupted, that is, one of the text modules referred to in the chain no longer exists.
NO_REFERENCE	In the text header, the fields TDREFOBJ, TDREFID, and TDREFNAME do not contain a reference text. One or more of these fields are empty.

SAVE_TEXT

SAVE_TEXT writes a text module back to the text file or the text memory, depending on the storage mode of the corresponding text object.

You can use this module either to change existing texts or to create new texts. If you know for sure that the text is new, use the parameter INSERT to indicate this. The system then does not have to read the text first, which improves the performance of the function module.

If the lines table passed with the function module is empty, the system deletes the text from the text file.

Function call:

```
CALL FUNCTION   'SAVE_TEXT'
     EXPORTING   CLIENT          = SY-MANDT
                 HEADER          = ?...
                 INSERT          = SPACE
                 SAVEMODE_DIRECT = SPACE
                 OWNER_SPECIFIED = SPACE
     IMPORTING   FUNCTION        =
                 NEWHEADER       =
     TABLES      LINES           = ?...
     EXCEPTIONS  ID              =
                 LANGUAGE        =
                 NAME            =
                 OBJECT          =
```

Export parameters:

CLIENT	Specify the client under which to store the text. If you omit this parameter, the system uses the current client as default. Reference field: SY-MANDT Default value: SY-MANDT
HEADER	Enter the structure that contains the text header of the text you want to save. Structure: THEAD

SAVE_TEXT

INSERT	This parameter indicates that the text module is new. Usually, SAPscript reads the text file first to check whether the text module exists. If you know that the text is new before the application program calls the function module, use this parameter to prevent the system from this read process and thus improve the performance. Possible values: ' ' Determine update mode automatically 'X' Text is new Default value: SPACE
SAVEMODE_DIRECT	You determine the storage mode of a text module (direct, in update task) via the text object in table TTXOB. However, it may be necessary to replace storage in update task with direct storage of a text (for example, for background processing). Possible values: ' ' Storage mode according to text object 'X' Save text module directly Default value: SPACE
OWNER_SPECIFIED	When creating a new text, the parameter indicates whether the creation information in the text header is filled in automatically by SAPscript, or whether to use the data passed in the header. The parameter concerns the fields TDFUSER, TDFDATE, TDFTIME, and TDFRELES in the text header. Possible values: ' ' Information taken from SAPscript 'X' Information taken from header If you use this parameter, but one of the required fields is initial, this field is filled by SAPscript. You use this parameter if you use a program to insert text under a certain owner or a certain release information, which do not correspond to the current environment (for example, a migration program for transferring texts from the R/2 system to the R/3 system). Default value: SPACE

Import parameters:

FUNCTION	The parameter returns the processing status of the text module for the current call. Possible values: ' ' no action 'I' text module was created 'U' text module was modified 'D' text module was deleted

Table parameters:

LINES	The table contains the text lines of the text to be saved. Structure: TLINE

Exceptions:

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The parameter LANGUAGE contains a language key that does not exist in table T002.
NAME	The parameter NAME contains the name of a text module that does not correspond to the SAPscript conventions. Possible errors: • The field contains only blanks. • The field contains the invalid characters '*' or ','.
OBJECT	The parameter OBJECT contains the name of a text object that does not exist in table TTXOB.

DELETE_TEXT

Use DELETE_TEXT to delete one or more text modules from the text file. You can enter generic values in the fields NAME, ID, and LANGUAGE. Depending on the corresponding text object, the system either deletes the texts directly or flags them in the text memory as to be deleted.

Function call:

```
CALL FUNCTION  'DELETE_TEXT'
     EXPORTING  CLIENT          = SY-MANDT

                OBJECT          = ?...

                NAME            = ?...

                ID              = ?...

                LANGUAGE        = ?...

                SAVEMODE_DIRECT = SPACE

                TEXTMEMORY_ONLY = SPACE

     EXCEPTIONS NOT_FOUND       =
```

Export parameters:

CLIENT	Specify the client in which the text you want to delete is stored. If you omit this parameter, the system uses the current client as default. This parameter is only valid for direct storage; it is not valid if you use the text memory. Reference field: SY-MANDT Default value: SY-MANDT
OBJECT	Enter the name of a text object. The system then deletes only those texts that are allocated to this text object. Reference field: THEAD-TDOBJECT
NAME	The parameter indicates the name of the text module to be deleted. You can enter a generic value. Reference field: THEAD-TDNAME
ID	Enter the text ID of the text you want to delete. You can enter a generic value. Reference field:THEAD-TDID

LANGUAGE	Enter the language key of the text module. You can enter a generic value. Reference field: THEAD-TDSPRAS
SAVEMODE_DIRECT	You determine the storage mode of a text module (direct, in update task) via the text object in table TTXOB. However, it may be necessary to replace storage in update task with direct storage of a text (for example, for background processing). Possible values: ' ' Storage mode according to text object 'X' Save text module directly Default value: SPACE
TEXTMEMORY_ONLY	Use this parameter to indicate that you want the delete function to apply for the text memory only. If the parameter contains 'X', the system does not delete the text itself from the text memory but only its entry. This allows you to rollback all changes made to a text during a transaction. The parameter is valid only for texts stored in the text memory.

Exceptions:

NOT_FOUND	The system did not find the specified text module.

COPY_TEXTS

Use this function module to copy texts. You can copy only texts that exist in the text file.

Apart from the text keys of the texts to be copied, COPY_TEXTS only needs the text keys of the target texts. You use table TEXTS to pass these values to the function module.

The system then copies the texts in blocks without first passing them to the text work areas.

After the function module is finished, you can read the result for each text in the field SUBRC of table TEXTS.

This function works much faster than the following copy procedure:

1. Read the text you want to copy using READ_TEXT.
2. Enter target text key in the header.
3. Save text module under the new name using SAVE_TEXT.

Function call:

```
CALL FUNCTION 'COPY_TEXTS'
     EXPORTING SAVEMODE_DIRECT = ' '
               INSERT          = ' '
     IMPORTING ERROR           =
     TABLES    TEXTS           = ?...
```

Export parameters:

SAVEMODE_DIRECT	You determine the storage mode of a text module (direct, in update task) via the text object in table TTXOB. However, it may be necessary to replace storage in update task with direct storage of a text (for example, for background processing). Possible values: ' ' Storage mode according to text object 'X' Save text module directly Default value: SPACE

INSERT	This parameter indicates that the text module is new. Usually, SAPscript reads the text file first to check whether the text module exists. If you know that the text is new before the application program calls the function module, use this parameter to prevent the system from this read process and thus improve the performance. Possible values: ' ' Determine update mode automatically 'X' Text is new Default value: SPACE

Import parameters:

ERROR	The parameter indicates whether an error occurred during the copy process. Possible values: ' ' no error 'X' error in copy process This error flag only shows that an error occurred during the copy process. For more information, read the field SUBRC in table TEXTS.

Table parameters:

TEXTS	The table contains the texts to be copied. For each table line, you can call a copy function. You must always specify the entire text keys of source and target texts. Generic values are not allowed. After finishing the copy process, the field SUBRC contains for each text the return value of the copy process. Possible values of SUBRC: 00 text copied 01 invalid target text ID 02 invalid characters in the target text name 03 invalid target text object 04 invalid storage mode for target text 05 error when reading source text header 06 error when reading target text header 07 error when writing target text header 08 error when copying text lines Structure: ITCTC

SELECT_TEXT

SELECT_TEXT creates a table from the text headers of all text modules that match the selections specified in the fields OBJECT, NAME, ID, and LANGUAGE. The entries in the fields OBJECT, NAME, ID, and LANGUAGE can also be generic.

Usually, the function module searches for texts in the text file as well as in the text memory. To limit the search area, use the options TEXTMEMORY_ONLY or DATABASE_ONLY.

If the option ARCHIV_HANDLE contains a value greater than 0, the system searches the archive for the specified texts.

Function call:

```
CALL FUNCTION  'SELECT_TEXT'
     EXPORTING  CLIENT                  = SY-MANDT
                OBJECT                  = ?...
                NAME                    = ?...
                ID                      = ?...
                LANGUAGE                = ?...
                DATABASE_ONLY           = SPACE
                TEXTMEMORY_ONLY         = SPACE
                ARCHIVE_HANDLE          = 0
     IMPORTING  ENTRIES                 =
     TABLES     SELECTIONS              = ?...
     EXCEPTIONS WRONG_ACCESS_TO_ARCHIVE =
```

Export parameters:

CLIENT	Specify the client, in which to search for the texts. If you omit this parameter, the system uses the current client as default. Reference field: SY-MANDT Default value: SY-MANDT
OBJECT	Enter the name of the text object of the text you want to find. You can enter a generic value. Reference field: THEAD-TDOBJECT

SELECT_TEXT

NAME	Enter the name of the text module you want to find. You can enter a generic value. Reference field: THEAD-TDNAME
ID	Enter a text ID. You can enter a generic value. Reference field: THEAD-TDID
LANGUAGE	Enter the language key of the text module you want to find. You can enter a generic value. The system then searches for the text in all languages that match the entry. Reference field: THEAD-TDSPRAS
DATABASE_ONLY	Mark this field if you want to search for the text modules in the text file only. Possible values: ' ' search in text file and text memory 'X' search in text file only Default value: SPACE
TEXTMEMORY_ONLY	Mark this field if you want to search for the text modules in the text memory only. Possible values: ' ' search in text file and in text memory 'X' search in text memory only Default value: SPACE
ARCHIVE_HANDLE	If you want the system to search for the text modules in the archive, you must enter a handle here. The system needs the handle to access the archive. You can use the function module **ACHIVE_OPEN_FOR_READ** to create the handle. The value '0' indicates that you do not want to read the archive. Reference field: SY-TABIX Default value: 0

Import parameters:

ENTRIES	ENTRIES contains the number of selected text modules. This value corresponds to the number of lines in table SELECTIONS. Reference field: SY-TFILL

Table parameters:

SELECTIONS	The table SELECTIONS contains the text headers of all found texts that match the selection criteria specified in the call of function module SELECT_TEXT. Structure: THEAD

Exceptions:

WRONG_ACCESS_TO_ARCHIVE	The exception WRONG_ACCESS_TO_ARCHIVE is triggered if an archive is accessed using an incorrect or non-existing archive handle or an incorrect mode (that is, read if the archive is open for writing or vice versa).

REFER_TEXT

You can create SAPscript texts as references to the text lines of other text modules. In this case, the system does not store text lines but only the references to the other text module. Changes to the reference text now affect the calling text module as well. You can create a reference chain over several levels.

The text module specified in the header refers to another text module. This text is specified using the parameters REF_OBJECT, REF_NAME, and REF_ID. The language of the reference text is always the same as the language of the calling text.

Usually, you will refer to a text that exists in the database. The system can then return the text lines of the reference text in the LINES table using READ_LINES. If the text exists in the text memory only and has not yet been written to the database, you can create the reference but not use READ_LINES to read the text lines of the reference text.

⚠

REFER_TEXT does not trigger a SAVE_TEXT function call. To save a reference text, you must explicitly call SAVE_TEXT after executing REFER_TEXT.

Handling in the Editor

In the fullscreen editor, you cannot edit the text lines of the reference text read into table LINES. The editor can only display them. If you want to make changes, you must unlock the text in the editor. In this case, however, the connection to the original reference text is lost.

The user can resolve the connection to the reference text in the editor by choosing *Text → Unlock*. The text lines are then ready for editing. However, any connection to the original reference text is lost.

To resolve the reference from within the program, simply delete the fields TDREFOBJ, TDREFNAME, and TDREFID in the text header.

Function call:

```
CALL FUNCTION  'REFER_TEXT'
     EXPORTING  HEADER          = ?...
                REF_OBJECT      = ?...
                REF_NAME        = ?...
                REF_ID          = ?...
                CHECK_REFERENCE = 'X'
                READ_LINES      = 'X'
     IMPORTING  NEWHEADER       =
     TABLES     LINES           = ?...
     EXCEPTIONS NOT_FOUND       =
            REFERENCE_CHECK =
```

Export parameters:

HEADER	Enter the text header of the text for which you want to create a reference to another text. Structure: THEAD
REF_OBJECT	The parameter contains the text object of the reference text. Reference field: THEAD-TDOBJECT
REF_NAME	Enter the name of the reference text. Reference field: THEAD-TDNAME
REF_ID	REF_ID contains the ID of the reference text. Reference field: THEAD-TDID
CHECK_REFERENCE	The parameter indicates whether you want the system to check the existence of the reference text. Possible values: ' ' no check 'X' the system checks whether the reference text exists. The system always executes the check if you want to read the lines of the reference text. Default value: 'X'
READ_LINES	Indicates whether to return the lines of the reference text in table LINES. Possible values: ' ' do not read text lines 'X' read text lines Default value: 'X'

Import parameters:

NEWHEADER	The field contains the modified header. The text header stores whether the text module refers to another text. Therefore, you must specify a corresponding structure; otherwise calling the function module REFER_TEXT has no effect. The structure is usually the same as the one used in the HEADER parameter. Structure: THEAD

Table parameters:

LINES	The table contains the text lines of the reference text, provided the parameter READ_LINES has the value 'X'. Structure: TLINE

Exceptions:

NOT_FOUND	The system did not find the specified text module.
REFERENCE_CHECK	The text module to be read has no text lines of its own but refers to the lines of another text module. This reference chain can include several levels. For the current text, the chain is interrupted, that is, one of the text modules referred to in the chain no longer exists.

RENAME_TEXT

In application programs, the complete name of a master record or document is known only directly before the COMMIT WORK. In the meantime, the system must use dummy names for the corresponding text modules. Before the COMMIT_TEXT, it must replace the dummy names with the final names, using the function module RENAME_TEXT. The text concerned receives the name specified in NEWNAME.

You can change only the name of one text module. OBJECT, ID, and LANGUAGE remain unchanged. Generic entries for ID and LANGUAGE are allowed.

⚠

The text module does not execute RENAME in the database. It only renames texts in the text memory.

Function call

```
CALL FUNCTION  'RENAME_TEXT'
     EXPORTING  OBJECT    = ?...
                NAME      = ?...
                ID        = ?...
                LANGUAGE  = ?...
                NEWNAME   = ?...
     EXCEPTIONS NOT_FOUND =
```

Export parameters

OBJECT	Enter the name of the text object to which the text is allocated. Reference field: THEAD-TDOBJECT
NAME	Enter the name of the text module you want to rename. Reference field: THEAD-TDNAME
ID	Enter the text ID of the text module you want to rename. You can enter a generic value. Reference field: THEAD-TDID
LANGUAGE	Enter the language key of the text module. You can enter a generic value. Reference field: THEAD-TDSPRAS
NEWNAME	NEWNAME determines the new text name. The system renames all texts in the text memory that match the selections specified in the parameters OBJECT, NAME, ID, and LANGUAGE. Reference field: THEAD-TDNAME

Exceptions

NOT_FOUND	The system did not find the specified text module.

COMMIT_TEXT

The system keeps all text modules for which you defined 'storage in update task' in the corresponding text object in the text memory. As soon as it updates the corresponding application object, it must also place the text modules into the log file.

The function module COMMIT_TEXT generates for the text modules in the text object a CALL FUNCTION... IN UPDATE TASK statement in accordance with the action to be executed (delete, create, change).

No COMMIT WORK is created. This must be executed by the application program.

If you do not specify OBJECT, NAME, ID, and LANGUAGE, the system transfers all texts from the text memory. To limit the function to certain texts, enter values (fully or generically) in the above fields. The system then selects all texts that match the selections in the fields up to the first '*'.

By default, the system deletes texts from the text memory as soon as they are written to the log file. If you want to keep updated texts in the text memory, call the function module with the parameter KEEP = 'X'. The system then keeps the texts in the text memory and flags them as updated. When calling COMMIT_TEXT again, the system ignores these texts. If you change such a text again during the transaction (for example, using SAVE_TEXT or DELETE_TEXT), the system deletes the flag. However, you need another COMMIT_TEXT to update the text.

Function call:

```
CALL FUNCTION  'COMMIT_TEXT'
     EXPORTING  OBJECT          = '*'
                NAME            = '*'
                ID              = '*'
                LANGUAGE        = '*'
                SAVEMODE_DIRECT = SPACE
                KEEP            = SPACE
      IMPORTING COMMIT_COUNT    =
```

Export parameters:

OBJECT	Enter the name of the text object. The system then transfers only texts with this text object to the log file. You can enter a generic value. If you omit this parameter, the system uses texts of all objects. Reference field: THEAD-TDOBJECT Default value: '*'
NAME	Enter the name of the text modules you want to update. You can enter a generic value. If you omit the parameter, the system uses all texts that match any other selections. Reference field: THEAD-TDNAME Default value: '*'
ID	Enter the text ID of the text modules. You can enter a generic value. If you omit this parameter, the system uses all IDs. Reference field: THEAD-TDID Default value: '*'
LANGUAGE	Enter the language key of the text modules. You can enter a generic value. If you omit this parameter, the system uses all languages. Reference field: THEAD-TDSPRAS Default value: '*'
SAVEMODE_DIRECT	You determine the storage mode of a text module (direct, in update task) via the text object in table TTXOB. However, it may be necessary to replace storage in update task with direct storage of a text (for example, for background processing). Possible values: ' ' Storage mode according to text object 'X' Save text module directly Default value: SPACE
KEEP	Use the parameter KEEP to indicate whether to keep updated texts in the text memory. This allows you to access these texts during the current transaction even though no COMMIT WORK was executed. If another COMMIT_TEXT occurs within the transaction, the texts kept with KEEP are ignored, unless you changed them again using function modules such as SAVE_TEXT or DELETE_TEXT. Default value: SPACE

Import parameters:

COMMIT_COUNT	The parameter returns the number of text modules transferred to the update task. Reference field: SY-INDEX

INIT_TEXT

For each text module you want to process using SAPscript, you must provide two work areas from within the application program: one for the text header and one for the lines table. You must initialize these work areas whenever you create a new text or the system terminated a function module READ_TEXT or READ_TEXT_INLINE with an exception.

You can reuse the work areas if the previously edited text is stored or if you no longer need it.

You must fully specify the text module in the fields OBJECT, NAME, ID, and LANGUAGE. Generic entries are not allowed.

After calling INIT_TEXT, the header contains initial values and the lines table is empty.

Function call:

```
CALL FUNCTION  'INIT_TEXT'
     EXPORTING  ID       = ?...
                LANGUAGE = ?...
                NAME     = ?...
                OBJECT   = ?...
     IMPORTING  HEADER   =
     TABLES     LINES    = ?...
     EXCEPTIONS ID       =
                LANGUAGE =
                NAME     =
                OBJECT   =
```

Export parameters:

OBJECT	Enter the name of the text object to which the text is allocated. Table TTXOB contains the valid objects. Reference field: THEAD-TDOBJECT
NAME	Enter the name of the text module. The name may be up to 70 characters long. Its internal structure depends on the text object used. Reference field: THEAD-TDNAME
ID	Enter the text ID of the text module. Table TTXID contains the valid text IDs, depending on the text object. Reference field: THEAD-TDID

LANGUAGE	Enter the language key of the text module. The system accepts only languages that are defined in table T002. Reference field: THEAD-TDSPRAS

Import parameters:

HEADER	This parameter returns the initialized text header. Structure: THEAD

Table parameters:

LINES	All lines of the specified text lines table are deleted. Structure: TLINE

Exceptions:

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The parameter LANGUAGE contains a language key that is not defined in table T002.
NAME	The parameter NAME contains a text module name that does not correspond to the SAPscript conventions. Possible errors: • Field contains only blanks • Field contains the invalid characters '*' or '.'
OBJECT	The parameter OBJECT contains the name of a text object that is not defined in table TTXOB.

EDIT_TEXT

This function module calls the fullscreen editor. You can use the editor functions to edit the text lines. The system sets the editor interface according to the interface specification in the text object.

Usually, the system implicitly calls the function module SAVE_TEXT if you leave the editor choosing *Save*, provided the text is stored in the text file according to the allocated text object. To deactivate this call, use the parameter SAVE.

If the field TDFORMAT in the text header contains SPACE, the system calls the SAPscript Editor. Otherwise, it calls the function module EDIT_TEXT_FORMAT_xxx where xxx is the contents of the field TDFORMAT. This function module then calls the editor required for the specified text format. Which of the parameters passed with EDIT_TEXT the system evaluates, depends on this interface module and the called word processing program.

Function call

```
      EXPORTING
                  DISPLAY      =  SPACE
                  EDITOR_TITLE=  SPACE
                  HEADER       =
                  SAVE         =  'X'
                  CONTROL      =  SPACE
                  PROGRAM      =  SPACE
      IMPORTING
                  NEWHEADER    =
                  FUNCTION     =
                  RESULT       =
      TABLES
                  LINES        =
      EXCEPTIONS
                  ID           =
                  LANGUAGE     =
                  LINESIZE     =
                  NAME         =
                  OBJECT       =
                  TEXTFORMAT   =
                  COMMUNICATION=
```

Export parameters

DISPLAY	The parameter indicates whether the text editor is called in display or in change mode. In display mode, the user cannot edit the text. Possible values: 'X' display mode ' ' change mode Default value: SPACE
EDITOR_TITLE	The system displays the title specified here in addition to the other headings generated by SAPscript in the title bar of the text editor window. Thus you can set any title text in the editor. This allows you to decrypt complicated technical text names to display them in the title bar. Reference field: TTXIT-TDTEXT Default value: SPACE
HEADER	Enter the structure that contains the text header of the text to be edited. Structure: THEAD
CONTROL	Use this parameter to set certain attributes of the SAPscript editor. The attributes are described in the fields of structure ITCED. Structure: ITCED Default value: SPACE
SAVE	Use this parameter to determine whether the system calls SAVE_TEXT after saving the text in the editor. If you want to suppress this call, the application program must later on call the function module SAVE_TEXT itself. In both cases, the system returns the changed text lines to the application program. Possible values: 'X' call SAVE_TEXT ' ' suppress SAVE_TEXT Default value: 'X'
PROGRAM	Use this parameter to determine the program whose work areas the system uses to replace the values of the program symbols. If you omit this parameter, the system uses the first program called to search for the field values (SY-CPROG). Reference field: SY-REPID Default value: SPACE

Import parameters

NEWHEADER	Return parameter for the text header. Due to certain operations in the editor also fields in the text header are changed. Structure: THEAD

FUNCTION	Return parameter for the processing status of the text module for the current editor call. Possible values: ' ' no action 'I' text module created 'U' text module changed 'D' text module deleted
RESULT	The SAPscript Editor uses this parameter to return status information to the calling program. The information is described by the fields of structure ITCER. Structure: ITCER

Table parameters

LINES	Contains a table with the lines of the text module. Structure: TLINE

Exceptions

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The language key specified in the parameter LANGUAGE of the text header is not defined in table T002.
LINESIZE	The field TDLINESIZE of the text header contains a line width less than 0 or greater than 132.
NAME	The parameter NAME of the text header contains the name of a text module, which does not correspond to the SAPscript conventions. Possible errors: • The field contains only blanks. • The field contains the invalid characters ‘*’ or ‘,’.
OBJECT	The parameter OBJECT in the text header contains the name of a text object, which is not defined in table TTXOB.

TEXTFORMAT	The text lines passed to the function module have a format, which cannot be processed by the current environment. The format is stored in the text header in the field TDTEXTTYPE. Possible reasons: • The text format is not supported. • The text format cannot be processed on the current front-end, since the required word processing program is not installed. • The text format cannot be processed under batch input conditions, since the required word processing program cannot be used for batch input. • The text format cannot be used in background processing, since the required word processing program cannot be called from within background processes.
COMMUNICATION	The field TDTEXTTYPE of the text header is not empty, that is, it contains a text format which requires an external word processing program to be called. When calling this program, an error occurred. Possible reasons: • The external word processing program is not or not correctly installed on the front-end. • Word processing could not be started. • During communication with this word processing program, an error occurred. • During text data transfer, an error occurred. When processing texts in a non-SAPscript format, the system internally calls the function modules EDIT_TEXT_FORMAT_xxx or PRINT_TEXT_FORMAT_xxx, where xxx is the contents of field TDTEXTTYPE. These function modules ended with the exception COMMUNICATION. SAPscript passes this message to the print program without any further analysis.

EDIT_TEXT_INLINE

In an application it may be necessary to display or edit the first few lines of a text in any screen. To do this, use the function module EDIT_TEXT_INLINE.

The system then merges the lines of table INLINES with those of table LINES. The parameter INLINE_COUNT determines the number of lines in table INLINES. If necessary, the user can branch to the fullscreen editor. In this case, the function module behaves like EDIT_TEXT.

After executing the function, the system fills table INLINES with the specified number of lines from LINES. You can use the parameter SAVE to specify whether to call the function module SAVE_TEXT automatically if any lines are changed.

The application program is responsible for outputting the text lines from table INLINES to the screen and for returning the changed lines after the user pressed ENTER. On the application screen, the user can use only the general editing functions provided for all screen fields.

The function module is called at the PBO event.

Function call:

```
CALL FUNCTION  'EDIT_TEXT_INLINE'
      EXPORTING
                  DISPLAY     = SPACE
                  EDITOR_TITLE= SPACE
                  HEADER      =
                  INLINE_COUNT=
                  SAVE        = 'X'
                  TEXTSCREEN  = SPACE
                  CONTROL     = SPACE
                  PROGRAM     = SPACE
      IMPORTING
                  FUNCTION    =
                  NEWHEADER   =
                  RESULT      =
      TABLES
                  INLINES     =
                  LINES       =
      EXCEPTIONS
                  ID          =
                  LANGUAGE    =
                  LINESIZE    =
                  NAME        =
                  OBJECT      =
```

Export parameters:

DISPLAY	The parameter indicates whether the text editor is called in display or in change mode. In display mode, the user cannot edit the text. Possible values: 'X' display mode ' ' change mode Default value: SPACE
EDITOR_TITLE	Enter a title that you want the system to display in the title bar of the text editor window in addition to the headings generated by SAPscript. Reference field: TTXIT-TDTEXT Default value: SPACE
HEADER	Enter the structure that contains the text header of the text to be edited. Structure: THEAD
INLINE_COUNT	Enter the number of text lines you want the system to pass from table LINES to table INLINES. The system always starts at the beginning of table LINES. This value corresponds to the number of text lines displayed on a screen.
SAVE	Use this parameter to determine whether the system calls SAVE_TEXT after saving the text in the editor. If you want to suppress this call, the application program must lateron call the function module SAVE_TEXT itself. In both cases, the system returns the changed text lines to the application program. Possible values: 'X' call SAVE_TEXT ' ' suppress SAVE_TEXT Default value: 'X'
TEXTSCREEN	The parameter determines whether, after comparing the INLINES lines, to call the text editor. Possible values: ' ' do not call text editor 'X' call text editor Default value: SPACE

CONTROL	Use the parameter CONTROL to set certain attributes of the SAPscript Editor. These attributes are described in the fields of structure ITCED. Structure: ITCED Default value: SPACE
PROGRAM	Use this parameter to determine the program whose work areas the system uses to replace the values of the program symbols. If you omit this parameter, the system uses the first program called to search for the field values (SY-CPROG). The form processor replaces program symbols in the print view only, but not in the editor itself. Reference field: SY-REPID Default value: SPACE

Import parameters:

FUNCTION	Return parameter for the processing status of the text module for the current editor call. Possible values: ' ' no action 'I' text module created 'U' text module changed 'D' text module deleted
NEWHEADER	Return parameter for the text header. Due to certain operations in the editor, also fields in the text header are changed. Structure: THEAD
RESULT	The SAPscript Editor uses this parameter to return status information to the calling program. The information is described by the fields of structure ITCER. Structure: ITCER

Table parameters:

INLINES	Contains the first INLINE_COUNT lines of table LINES. The application program must transfer these lines at the PAI event to the corresponding fields of the appropriate application screen. And, at the PBO event, it must return the lines from the screen fields to the table before calling the function module. Structure: TLINE

LINES	Contains the table with the lines of the text module. Structure: TLINE

Exceptions:

ID	The text ID specified in the parameter ID does not exist in table TTXID. It must be defined there together with the object of the text module.
LANGUAGE	The language key specified in the parameter LANGUAGE of the text header is not defined in table T002.
LINESIZE	The field TDLINESIZE of the text header contains a line width less than 0 or greater than 132.
NAME	The parameter NAME or the field TDNAME of the text header contains the name of a text module, which does not correspond to the SAPscript conventions. Possible errors: • The field contains only blanks. • The field contains the invalid characters '*' or ','.
OBJECT	The parameter OBJECT in the text header contains the name of a text object, which is not defined in table TTXOB.

CHECK_TEXT_AUTHORITY

The function module checks whether the user is authorized to display or change the specified text module. You can use this function for texts of object type TEXT only. These are the standard texts that you edit using transaction SO10. If the user has no authorization, the system terminates the function module with the exception NO_AUTHORITY. The system checks the user authorization in the authorization object S_SCRP_TXT.

Function call:

```
CALL FUNCTION  'CHECK_TEXT_AUTHORITY'
     EXPORTING  OBJECT        = 'TEXT      '
                NAME          = ?...
                ID            = ?...
                LANGUAGE      = ?...
                ACTIVITY      = ?...
     EXCEPTIONS NO_AUTHORITY =
```

Export parameters:

OBJECT	Enter the name of the text object allocated to the text. At present, only TEXT (standard texts) is allowed. Reference field: THEAD-TDOBJECT
NAME	Enter the name of the text module for which to execute the check. Reference field: THEAD-TDNAME
ID	Enter the ID of this text. Reference field: THEAD-TDID
LANGUAGE	Enter the language key of this text. Reference field: THEAD-TDSPRAS
ACTIVITY	In the parameter ACTIVITY, you must specify the functions used to edit the text module in the application program. Possible values: ‘SHOW’display the test module 'EDIT' change the text module

Exceptions:

NO_AUTHORITY	The user is not authorized to edit the specified standard text (object TEXT) using the chosen function. The system checks the authorization against the object S_SCRP_TXT with the fields text name, text ID within the text object TEXT, text language, and chosen activity. Possible reasons: • The parameter ACTIVITY contains the value ‘SHOW’ (display). The user is not authorized to display the specified text module, that is in the field ACTVT, the authorization value ‘03’ is missing for the specified text. • The parameter ACTIVITY contains the value ‘EDIT’ (change). The user is not authorized to change the specified text module, that is in the field ACTVT, the authorization value ‘02’ is missing for the specified text. If non of these reasons is the cause of the error, the reason could be: • The field ACTIVITY contains an invalid value. • Error when calling the authorization check. For information on this error, see the documentation of the ABAP statement AUTHORITY-CHECK.

CHECK_TEXT_ID

CHECK_TEXT_ID checks whether the specified text ID is defined in table TTXID. If it exists, the parameter ID_INFO contains the corresponding entry in table TTXID. If it does not exist, the system terminates the function module with the exception ID.

Function call:

```
CALL FUNCTION  'CHECK_TEXT_ID'
     EXPORTING  OBJECT   = ?...
                ID       = ?...
     IMPORTING  ID_INFO  =
     EXCEPTIONS ID       =
```

Export parameters:

OBJECT	The parameter OBJECT contains the text object that belongs to the text ID. Reference field: THEAD-TDOBJECT
ID	The parameter ID contains the text ID to be checked. Reference field: THEAD-TDID

Import parameters:

ID_INFO	Contains the attributes defined in table TTXID for the specified text ID. Reference field: THEAD-TDID

Exceptions:

ID	The text ID specified in the parameter ID or in the field TDID of the text header does not exist in table TTXID. It must be defined there together with the object of the text module.

CHECK_TEXT_LANGUAGE

CHECK_TEXT_LANGUAGE checks whether the language of a text module is defined in table T002. If not, the system terminates the function module with the exception LANGUAGE.

Function call:

```
CALL FUNCTION  'CHECK_TEXT_LANGUAGE'
     EXPORTING  LANGUAGE = ?...
     EXCEPTIONS LANGUAGE =
```

Export parameters:

LANGUAGE	The parameter LANGUAGE contains the text language to be checked. Reference field: THEAD-TDSPRAS

Exceptions:

LANGUAGE	The parameter LANGUAGE contains a text language key that is not defined in table T002.

CHECK_TEXT_OBJECT

CHECK_TEXT_OBJECT checks whether the specified text object is defined in table TTXOB. If yes, the parameter OBJECT_INFO contains the corresponding entry of table TTXOB. If no, the system terminates the function module with the exception OBJECT.

Function call:

```
CALL FUNCTION  'CHECK_TEXT_OBJECT'
     EXPORTING  OBJECT      = ?...
     IMPORTING  OBJECT_INFO =
     EXCEPTIONS OBJECT      =
```

Export parameters:

OBJECT	The parameter OBJECT contains the text object to be checked. Reference field: THEAD-TDOBJECT

Import parameters:

OBJECT_INFO	Contains the attributes defined in table TTXOB for the specified text object. Reference field: THEAD-TDOBJECT

Exceptions:

OBJECT	The parameter OBJECT or the field TDOBJECT in the text header contain the name of a text object that is not defined in table TTXOB.

CHECK_TEXT_NAME

CHECK_TEXT_NAME checks whether the name of a text module consists of valid characters only. Valid characters are all characters of the system character set except the characters `'*'` and `','`. If one of these characters occurs in the field specified in the parameter NAME, the system terminates the function module with the exception NAME.

Function call:

```
CALL FUNCTION  'CHECK_TEXT_NAME'
     EXPORTING  NAME = ?...
     EXCEPTIONS NAME =
```

Export parameters:

NAME	The parameter NAME contains the text name to be checked. Reference field: THEAD-TDNAME

Exceptions:

NAME	The parameter NAME or the field TDNAME of the text header contain the name of a text module that does not correspond to the SAPscript conventions. Possible errors: • Field contains only blanks • Field contains the invalid characters '*' or ','

TEXT_SYMBOL_COLLECT

This function module returns a table that contains all symbols used in the text lines, which are transferred in the table parameter LINES. For each symbol, the table specifies how often it occurs in the text.

The table is sorted in ascending order by symbol names.

Function call:

```
CALL FUNCTION 'TEXT_SYMBOL_COLLECT'
     TABLES     LINES   = ?...
                SYMBOLS = ?...
```

Table parameters:

LINES	Enter the table whose text lines you want to search for symbols. Structure: TLINE
SYMBOLS	The table contains the names of all symbols used in the specified text. The system enters these names without the escape symbol &. Structure: ITCST

TEXT_SYMBOL_PARSE

This function module analyses whether a character string in SAPscript format within a text line is a SAPscript symbol.

Pass the text line you want to analyze in LINE. The parameter START_OFFSET specifies the position of the start symbol & that is required for SAPscript symbols.

The system analyses the character string. If it is a symbol, it returns all information on this symbol in the parameters SYMBOL_....

If it is no SAPscript symbol, the system returns the exception NO_SYMBOL.

Function call:

```
CALL FUNCTION  'TEXT_SYMBOL_PARSE'
     EXPORTING  LINE                       = ?...
                START_OFFSET               = ?...
     IMPORTING  CONTINUE_OFFSET            =
                SYMBOL_CONDENSED           =
                SYMBOL_DECIMALS            =
                SYMBOL_DICTLEN             =
                SYMBOL_EXPONENT            =
                SYMBOL_FILLCHAR            =
                SYMBOL_LDATE               =
                SYMBOL_LENGTH              =
                SYMBOL_NAME                =
                SYMBOL_NOCONVERT           =
                SYMBOL_NOINIT              =
                SYMBOL_NOSIGN              =
                SYMBOL_NOZERO              =
                SYMBOL_OFFSET              =
                SYMBOL_RIGHT               =
                SYMBOL_SIGNLEFT            =
                SYMBOL_SIGNRIGHT           =
                SYMBOL_TEXT1               =
                SYMBOL_TEXT1_LENGTH        =
                SYMBOL_TEXT2               =
                SYMBOL_TEXT2_LENGTH        =
                SYMBOL_SEPARATOR_THOUSAND =
                SYMBOL_INCREMENT          =
                SYMBOL_DECREMENT          =
     EXCEPTIONS NO_SYMBOL                 =
```

Export parameters:

LINE	Contains the text line in SAPscript format (structure TLINE), whose contents you want to analyze. Structure: TLINE
START_OFFSET	Define the offset with reference to TLINE, where the character string to be analyzed starts. The offset must point to the start symbol & of the possible symbol.

Import parameters:

CONTINUE_OFFSET	CONTINUE_OFFSET points to the first character after the SAPscript symbol, provided it is a syntactically correct symbol. If the value is >= 134, the end symbol & of the symbol is positioned at the end of the line.
SYMBOL_CONDENSED	The parameter specifies whether for the analyzed symbol the formatting option C is set (compress blanks). Possible values: 'X' option set ' ' option not set
SYMBOL_DECIMALS	The parameter returns the number of decimal places set as option. If the parameter is empty, the option was not set for the analyzed symbol.
SYMBOL_DICTLEN	The parameter specifies whether the formatting option * is set for the analyzed symbol (output length according to Dictionary definition). Possible values: 'X' option set ' ' option not set

SYMBOL_EXPONENT	The parameter returns the exponent specified as option. If the parameter is empty, the option was not set for the analyzed symbol. If only the option E without any further digits was specified, the parameter contains the return value 0.
SYMBOL_FILLCHAR	The parameter specifies whether for the analyzed symbol the formatting option F is set (fill character for leading blanks). Possible values: ' ' no fill character or fill character SPACE other specified fill character
SYMBOL_LDATE	The parameter specifies whether for the analyzed symbol the formatting option L is set (local date evaluation). Possible values: 'X' option set ' ' option not set
SYMBOL_LENGTH	The parameter returns the length set as option. If the parameter is empty, no length was specified for the analyzed symbol.
SYMBOL_NAME	The parameter contains the name of the symbol without escape symbol &. Independent of the case used in the text line, the system always returns the name in uppercase.
SYMBOL_NOCONVERT	The parameter specifies whether for the analyzed symbol the formatting option K is set (ignore conversion routine from Dictionary). Possible values: 'X' option set ' ' option not set
SYMBOL_NOINIT	The parameter specifies whether for the analyzed symbol the formatting option I is set (suppress initial value). Possible values: 'X' option set ' ' option not set
SYMBOL_NOSIGN	The parameter specifies whether for the analyzed symbol the formatting option S is set (suppress sign). Possible values: 'X' option set ' ' option not set

SYMBOL_NOZERO	The parameter specifies whether for the analyzed symbol the formatting option Z is set (suppress leading zeroes). Possible values: 'X' option set ' ' option not set
SYMBOL_OFFSET	The parameter returns the offset set as option. If the parameter is empty, no offset was specified for the analyzed symbol.
SYMBOL_RIGHT	The parameter specifies whether for the analyzed symbol the formatting option R is set (output right-justified). Possible values: 'X' option set ' ' option not set
SYMBOL_SIGNLEFT	The parameter specifies whether for the analyzed symbol the formatting option < is set (sign on the left). Possible values: 'X' option set ' ' option not set
SYMBOL_SIGNRIGHT	The parameter specifies whether for the analyzed symbol the formatting option > is set (sign on the right). Possible values: 'X' option set ' ' option not set
SYMBOL_TEXT1	The parameter specifies which prefix text is specified for the analyzed symbol. You must interpret the contents of the parameter in connection with the parameter SYMBOL_TEXT1_LENGTH.
SYMBOL_TEXT1_LENGTH	The parameter specifies the length of the prefix text. The contents is specified in the parameter SYMBOL_TEXT1. If the value is 0, no prefix text is specified.
SYMBOL_TEXT2	The parameter specifies which suffix text is specified for the analyzed symbol. You must interpret the contents of the parameter in connection with the parameter SYMBOL_TEXT2_LENGTH.
SYMBOL_TEXT2_LENGTH	The parameter specifies the length of the suffix text. The contents is specified in the parameter SYMBOL_TEXT2. If the value is 0, no suffix text is specified.

SYMBOL_SEPARATOR_ THOUSAND	The parameter specifies whether for the analyzed symbol the formatting option T is set (no separator between thousands). Possible values: 'X' option set ' ' option not set
SYMBOL_INCREMENT	The parameter specifies whether for the analyzed symbol the formatting option '+' is set. This means that before outputting the value the system increases the value of the corresponding counter SAPSCRIPT-COUNTER_x (x = 0.. 9) by 1. Possible values: 'X' option set ' ' option not set
SYMBOL_DECREMENT	The parameter specifies whether for the analyzed symbol the formatting option '-' is set. This means that before outputting the value the system decreases the value of the corresponding counter SAPSCRIPT-COUNTER_x (x = 0.. 9) by 1. Possible values: 'X' option set ' ' option not set

Exceptions:

NO_SYMBOL	The character string that starts at the specified position is not a symbol of the SAPscript syntax. Possible errors: • The symbol was not closed by the character ‘&’. • The name of the symbol contains blanks. • The symbol extends over the end of a SAPscript editor line. • Additional formatting options are not enclosed in parentheses. • The key letters of the formatting options are not written in uppercase. • An invalid formatting option was entered. • The offset specification does not follow the symbol name directly. • Previous and/or subsequent texts are not enclosed in inverted commas.

TEXT_SYMBOL_REPLACE

The function module replaces the symbols contained in the text lines with their respective values. The text is not formatted for printing, but the symbols are replaced in the ITF format. Symbols that occur on comment or raw lines remain unchanged, as well as symbols enclosed by the character formats <(> and <)>.

The system interprets DEFINE statements in text lines and changes the values of text symbols accordingly. It also executes the control statements SET DATE MASK, SET TIME MASK, SET SIGN LEFT, and SET SIGN RIGHT.

Function call:

```
CALL FUNCTION 'TEXT_SYMBOL_REPLACE'
      EXPORTING
                  ENDLINE            = 99999
                  HEADER             =
                  INIT               = ' '
                  OPTION_DIALOG      = ' '
                  PROGRAM            = SPACE
                  REPLACE_PROGRAM    = 'X'
                  REPLACE_STANDARD   = 'X'
                  REPLACE_SYSTEM     = 'X'
                  REPLACE_TEXT       = 'X'
                  STARTLINE          = 1
            IMPORTING
                  CHANGED            =
                  NEWHEADER          =
      TABLES
                  LINES              =
```

Export parameters:

TEXT_SYMBOL_REPLACE

ENDLINE	Enter the index of the last text line on which to execute the function module. If you omit the parameter or specify an invalid value, the system ends with the last text table line. Reference field: SY-TABIX Default value: 99999
HEADER	The parameter contains the header of the text module whose symbols you want to replace. Structure: THEAD
INIT	Use INIT to specify whether you want to initialize the symbol administration of SAPscript before replacing. Possible values: 'X' initialize ' ' do not initialize Default value: ' '
OPTION_DIALOG	The parameter determines whether to display a dialog box before replacing, where the user can select the symbol types to be replaced. Possible values: 'X' display dialog box ' ' do not display dialog box Default value: ' '
PROGRAM	Use this parameter to determine the program whose work areas the system uses to replace the values of the program symbols. If you omit this parameter, the system uses the first program called to search for the field values (SY-CPROG). This assignment is valid for the current call of the funtion module only. Reference field: SY-REPID Default value: SPACE
REPLACE_PROGRAM	The parameter specifies whether to resolve program symbols during replacing. Possible values: 'X' replace program symbols ' ' do not replace program symbols Default value: 'X'

REPLACE_STANDARD	This parameter specifies whether to resolve standard symbols during replacing. Possible values: 'X' replace standard symbols ' ' do not replace standard symbols Default value: 'X'
REPLACE_SYSTEM	The parameter specifies whether to resolve system symbols during replacing. The system symbols &PAGE& and &NEXTPAGE& are not replaced, since the text is not formatted for printing. Possible values: 'X' replace system symbols ' ' do not replace system symbols Default value: 'X'
REPLACE_TEXT	The parameter specifies whether to resolve text symbols during replacing. Possible values: 'X' replace text symbols ' ' do not replace text symbols Default value: 'X'
STARTLINE	Enter the index of the table line from which to start the function module. If you omit the parameter or specify an invalid value, the system starts on the first text table line. Reference field: SY-TABIX Default value: 1

Import parameters:

CHANGED	The parameter indicates whether the symbols were replaced with their respective values, thus changing the contents of the text table. Possible values: 'X' symbols replaced ' ' symbols not replaced
NEWHEADER	The parameter returns the text header with the fields changed according to the executed action. Structure: THEAD

Table parameters:

LINES	The table contains the text lines, in which to replace the SAPscript symbols. Structure: TLINE

TEXT_SYMBOL_SETVALUE

Use the function module TEXT_SYMBOL_SETVALUE to assign a value to a SAPscript symbol. However, you can change only the values of text symbols. System symbols, standard symbols, and program symbols cannot receive new values.

The system keeps the new value until you initialize the symbol administration tables in SAPscript. To do this, you can use, for example, OPEN_FORM or TEXT_SYMBOL_REPLACE with the parameter INIT = ´X´.

Function call:

```
CALL FUNCTION 'TEXT_SYMBOL_SETVALUE'
     EXPORTING NAME            = ?...
               VALUE           = ?...
               VALUE_LENGTH    = 0
               REPLACE_SYMBOLS = ' '
```

Export parameters:

NAME	The parameter contains the name of the symbol to which you want to assign a new value. You must enter only the symbol name, without the texts or any formatting options. You can include the escape symbols &...&.
VALUE	Define the value of the symbol. The system can use up to 80 characters as symbol value.
VALUE_LENGTH	If you omit this parameter or specify 0, the system automatically determines the length of the symbol value. The system then includes all characters up to the last blank into the value. If you want to include trailing blanks into the assignment, specify the desired length of the symbol value here. Reference field: SY-TABIX Default value: 0

TEXT_SYMBOL_SETVALUE

REPLACE_SYMBOLS	The value 'X' in this parameter means that symbols contained in the field VALUE are replaced immediately with their current values; the assigned value no longer contains SAPscript symbols. If the parameter is empty, the system includes the contents of the VALUE field without change. Any symbols occurring within are replaced with their current value only after the symbol specified in the parameter NAME is called in the text. Default value: ' '

TEXT_CONTROL_REPLACE

The system interprets the statement lines IF, ELSE, ELSEIF, ENDIF, CASE, WHEN, OTHERS, ENDCASE in the transferred text table LINES and returns the resulting text into the table.

Function call:

```
CALL FUNCTION 'TEXT_CONTROL_REPLACE'
     EXPORTING HEADER          = ?...
               PROGRAM         = SPACE
               REPLACE_COMMENT = 'X'
     IMPORTING NEWHEADER       =
               CHANGED         =
     TABLES    LINES           = ?...
```

Export parameters:

HEADER	The parameter contains the header of the text module whose control statements you want the system to interpret. Structure: THEAD
PROGRAM	When interpreting control statements, SAPscript must determine the values of program symbols. To do this, it must know which active program contains the work areas for the values to be passed. If you omit the parameter, the system searches for the field values in the program that was called first (SY-CPROG). If you enter a program name, the system replaces the program symbols with the values from this program. This applies only for the current call of the function module. Reference field: SY-REPID Default value: SPACE
REPLACE_COMMENT	Use this parameter to determine whether to delete any comment lines from the text lines. Possible values: 'X' delete comment lines ' ' keep comment lines Default value: 'X'

Import parameters:

NEWHEADER	The parameter returns the text header with the fields changed according to the executed action. Structure: THEAD
CHANGED	The parameter indicates whether the system interpreted control statements and thus changed the contents of the text table. Possible values: 'X' statements interpreted ' ' no statements found

Table parameters:

LINES	The table contains the text lines in which to resolve the SAPscript statements. Structure: TLINE

TEXT_INCLUDE_REPLACE

Use this function module to resolve the INCLUDE statements that occur in the text lines, that is, to replace them with the respective lines of the text to be included. If a text module does not exist or if the user has no authorization, the system keeps the INCLUDE statement.

Function call:

```
CALL FUNCTION 'TEXT_INCLUDE_REPLACE'
     EXPORTING HEADER      = ?...
               STARTLINE   = 1
               ENDLINE     = 99999
               PROGRAM     = SPACE
               ALL_LEVEL   = 'X'
     IMPORTING NEWHEADER   =
               CHANGED     =
               ERROR_TYPE  =
     TABLES    LINES       = ?...
```

Export parameters:

HEADER	The parameter contains the header of the text module whose INCLUDE statements you want to resolve. Structure: THEAD
STARTLINE	Enter the index of the table line where you want the function module to start. If you omit the parameter or enter an invalid value, the system starts from the first table line. Reference field: SY-TABIX Default value: 1
ENDLINE	Enter the index of the table line where you want the function module to end. If you omit the parameter or enter an invalid value, the system ends with the last table line. Reference field: SY-TABIX Default value: 99999

PROGRAM	When interpreting control statements, SAPscript must determine the values of program symbols. To do this, it must know which active program contains the work areas for the values to be passed. If you omit the parameter, the system searches for the field values in the program that was called first (SY-CPROG). If you enter a program name, the system replaces the program symbols with the values from this program. This applies only for the current call of the function module. Reference field: SY-REPID Default value: SPACE
ALL_LEVEL	ALL_LEVEL determines whether to resolve only the INCLUDE statements in the text table or whether to resolve any new INCLUDE statements, which appear after resolving, as well. Possible values: ' ' resolve only existing INCLUDEs 'X' resolve all INCLUDEs Default value: 'X'

Import parameters:

NEWHEADER	The parameter returns the text header with the fields changed according to the executed action. Structure: THEAD
CHANGED	The parameter indicates whether the system resolved INCLUDEs and thus changed the contents of the text table. Possible values: 'X' INCLUDEs resolved ' ' no INCLUDEs resolved
ERROR_TYPE	At present, the parameter does not yet return error messages. Reference field: SY-TABIX

Table parameters:

LINES	The table contains the text lines in which to resolve the INCLUDE statements. Structure: TLINE

PRINT_TEXT

Use this function module to prepare the text module specified in the parameters HEADER and LINES for an output device and to output it. The system takes the required style and form specifications from the fields TDSTYLE or TDFORM of the text header.

Use the parameter OPTIONS to set different formatting and printing options. You specify the options in a structure like ITCPO by entering the desired values. The user can change some of these options on the optional print parameter screen (parameter DIALOG). This print parameter screen also appears, if no or invalid values for the required print options are specified in the parameter OPTIONS or the user master record.

Before printing a text, you can display it on the screen in the print format (from within the program, use field TDPREVIEW in the parameter OPTIONS; or the user can explicitly call the function on the print control screen).

If the field TDFORMAT in the text header contains SPACE, the SAPscript composer prepares the text for output by calling the function module PRINT_TEXT_FORMAT_xxx, where xxx is the contents of field TDFORMAT. This function module then calls the word processing program designed for that text format to prepare the text for printing. Which of the parameters passed with PRINT_TEXT the system finally evaluates, depends on the interface function module and on the word processing program.

Function call:

```
CALL FUNCTION  'PRINT_TEXT'
     EXPORTING  HEADER             = ?...
                DEVICE             = 'PRINTER'
                DIALOG             = 'X'
                OPTIONS            = SPACE
                APPLICATION        = 'TX'
                ARCHIVE_INDEX      = SPACE
                ARCHIVE_PARAMS     = SPACE
     IMPORTING  RESULT             =
                NEW_ARCHIVE_PARAMS =
     TABLES     LINES              = ?...
                OTFDATA            = ?...
     EXCEPTIONS CANCELED           =
                DEVICE             =
                FORM               =
                OPTIONS            =
                UNCLOSED           =
                UNKNOWN            =
                FORMAT             =
                TEXTFORMAT         =
                COMMUNICATION      =
```

Export parameters:

HEADER	The parameter contains the header of the text module which you want to prepare for printing. For print formatting, the system uses only the contents of the fields TDSTYLE and TDFORM of the header. Structure: THEAD
DEVICE	SAPscript can format a text for output on different device types. Enter the desired device type here. Possible values: 'PRINTER' print output 'TELEX' telex output 'TELEFAX' telefax output 'ABAP' screen output as ABAP list (interface of the calling program) 'SCREEN' screen output as ABAP list (interface controlled by SAPscript, can be set with parameter APPLICATION) The user can display output formatted for PRINTER, TELEX, or TELEFAX as print view on the screen. From within the program, you can set the field TDPREVIEW in the parameter OPTIONS, or the user can call the function on the print control screen. Default value: 'PRINTER'
DIALOG	Use parameter DIALOG to determine whether to display a dialog box before printing, in which the user can set several spool parameters for print formatting. Possible values: ' ' display no print parameter screen 'X' display print parameter screen Default value: 'X'

OPTIONS	Use parameter OPTIONS to set several options for print formatting. The parameter has the structure ITCPO. The user can change some of the defined settings on the print control screen. Structure: ITCPO Default value: SPACE
APPLICATION	For the device type SCREEN, the system displays the text formatting on the screen. This requires an interface in which the different menu entries are defined. The same applies if the user chooses to display a print view on the screen for the other device types. Enter one of the interface names provided by SAPscript. You usually use the interface that is assigned to the respective text object in table TTXOB. Reference field: TTXOB-TDAPPL Default value: 'TX'
ARCHIVE_INDEX	Enter the index information for the print output you want to archive. This information (DARA line) is stored in the archive together with the print output. Thus, you can use the index information to access the print output directly in the archive. Structure: TOA_DARA Default value: SPACE
ARCHIVE_PARAMS	The system interprets the settings passed in this parameter when archiving the output. The archive parameters have the ABAP Dictionary structure ARC_PARAMS. Structure: ARC_PARAMS Default value: SPACE

Import parameters:

RESULT	The parameter contains results of the print formatting process. By comparing the corresponding fields of parameter OPTIONS with those of parameter RESULT, you can determine whether the user made changes to any settings on the print control screen. Structure: ITCPP
NEW_ARCHIVE_PARAMS	The parameter contains results of the archiving process, including the archive parameters the user changed on the print control screen. The parameter has the ABAP Dictionary structure ARC_PARAMS. Structure: ARC_PARAMS

Table parameters:

LINES	Enter the table that contains the text lines you want to print. Structure: TLINE
OTFDATA	If in the parameter OPTIONS the field TDGETOTF contains 'X', the optional table parameter OTFDATA returns the formatted output in the OTF format. In this case, the system does not output the text on printer, screen or fax/telex/teletex devices. Structure: ITCOO

Exceptions:

CANCELED	When starting SAPscript print formatting, the system displayed a selection screen, on which the user can enter settings for the output, such as: • Printer name • Information on the cover page • Page selection • Number of copies The subsequent actions allowed on this screen were not called, but the output formatting was canceled instead. The function module was ended without further action. No form is open for output anymore.
DEVICE	The parameter DEVICE contains an invalid device type.
FORM	The parameter FORM contains the name of a form that the system could not find. Possible reasons: • The form does not exist. • There is no active version of this form. SAPscript first searches for the form in the current client and in the specified language. If the form does not exist there, it tries the original language of the form. If the form is still not found, it searches in client 0, first in the specified language, than in the original form language.
FORMAT	Within a print output defined between OPEN_FORM and CLOSE_FORM, you must use forms of the same page format. The page format of the current form differs from that of the previously called forms. Include only those forms into one spool request that have the same format (for example, DINA4). Page orientation is of no consequence; that means, you can mix pages in landscape and portrait format.

OPTIONS	The parameter OPTIONS contains invalid values for the formatting options. Possible errors: • The output device specified in field TDDEST does not exist. • The field TDPAGESLCT for selecting the pages to be printed contains invalid characters.
UNCLOSED	The system was told to open a new form even though an old form is still active. The old form must be closed first (CLOSE_FORM or END_FORM).
TEXTFORMAT	The text lines passed to the function module have a format that cannot be processed by the current environment. The format is stored in the text header in the field TDTEXTTYPE. Possible reasons: • The text format is not supported. • The text format cannot be processed on the current front-end, since the required word processing program is not installed. • The text format cannot be processed under batch input conditions, since the required word processing program cannot be used for batch input. • The text format cannot be used in background processing, since the required word processing program cannot be called from within background processes.
COMMUNICATION	The field TDTEXTTYPE of the text header is not empty, that is, it contains a text format which requires an external word processing program to be called. When calling this program, an error occurred. Possible reasons: • The external word processing program is not or not correctly installed on the front-end. • Word processing could not be started. • During communication with this word processing program, an error occurred. • During text data transfer, an error occurred. When processing texts in a non-SAPscript format, the system internally calls the function modules EDIT_TEXT_FORMAT_xxx or PRINT_TEXT_FORMAT_xxx, where xxx is the contents of field TDTEXTTYPE. These function modules ended with the exception COMMUNICATION. SAPscript passes this message to the print program without any further analysis.

UNKNOWN	An unknown error occurred. This exception is used only by the function module PRINT_TEXT, which, at former releases, passed all errors using this one exception. In the meantime, the exceptions of PRINT_TEXT were enhanced, so that the relevant error messages are covered by the module's own exceptions.

PRINT_TEXT_ITF

The function module prints the SAPscript text passed in table LINES in ITF format, which means that it prints the text lines in the ways they appear in the lines table. The printout thus resembles the representation in the SAPscript Editor.

The system outputs the text into the form SAPSCRIPT_ITF. The form can output up to 76 characters per line. This means that the contents of a SAPscript text line fits into such a form line only up to column 72. If a text line is longer, the system splits it in two. In the format line, the system inserts the '..' character to indicate that this line contains the columns 73 to 132 of the previous text line.

Function call:

```
CALL FUNCTION 'PRINT_TEXT_ITF'
     EXPORTING HEADER  = ?...
               OPTIONS = ?...
     IMPORTING RESULT  =
     TABLES    LINES   = ?...
```

Export parameters:

HEADER	The parameter contains the header of the text module whose lines you want to print. Structure: THEAD
OPTIONS	Use parameter OPTIONS to set different options for print formatting. The parameter has the structure ITCPO. The user can change some of the defined settings on the print control screen. Structure: ITCPO Default value: SPACE

Import parameters:

RESULT	The parameter contains results of the print formatting process. Structure: ITCPP

Table parameters:

LINES	Enter the table that contains the text lines you want to print. Structure: TLINE

OPEN_FORM

The function module OPEN_FORM opens form printing. You must call this function module before you can use any other form function (WRITE_FORM, START_FORM, CONTROL_FORM...).

You need not specify a form name. If you omit the name, you must use the function module START_FORM to open a form before starting the output.

You must end form printing by using the function module CLOSE_FORM. Otherwise, the system does not print or display anything.

Within a program, you can use several OPEN_FORM.. CLOSE_FORM pairs. This allows you to write output to several different spool requests from within one program.

Function call:

```
CALL FUNCTION  'OPEN_FORM'
     EXPORTING  FORM                = SPACE
                LANGUAGE            = SY-LANGU
                DEVICE              = 'PRINTER'
                DIALOG              = 'X'
                OPTIONS             = SPACE
                APPLICATION         = 'TX'
                ARCHIVE_INDEX       = SPACE
                ARCHIVE_PARAMS      = SPACE
     IMPORTING  LANGUAGE            =
                RESULT              =
                NEW_ARCHIVE_PARAMS  =
     EXCEPTIONS CANCELED            =
                DEVICE              =
                FORM                =
                OPTIONS             =
                UNCLOSED            =
```

Export parameters:

FORM	You can enter the name of a form here, which then controls output formatting. After calling OPEN_FORM, you can immediately output texts to the form using other function modules. If you leave the parameter blank, you must call START_FORM with a valid form name before starting any output functions. Default value: SPACE
LANGUAGE	Forms are language-dependent. Enter the desired language. If a form does not exist in this language, the system tries to call the form in its original language. Reference field: THEAD-TDSPRAS Default value: SY-LANGU
DEVICE	SAPscript can format a text for output on different device types. Enter the desired device type here. Possible values: 'PRINTER' print output 'TELEX' telex output 'TELEFAX' telefax output 'ABAP' screen output as ABAP list (interface of the calling program) 'SCREEN' screen output as ABAP list (interface controlled by SAPscript, can be set with parameter APPLICATION) The user can display output formatted for PRINTER, TELEX, or TELEFAX as print view on the screen. From within the program, you can set the field TDPREVIEW (structure ITCPO) in the parameter OPTIONS, or the user can call the function on the print control screen. The fields TDSENDTIME and TDSENDDATE designed for fax output (structure ITCPO) will be used for future enhancements; they are not used at present. Default value: 'PRINTER'
DIALOG	Use parameter DIALOG to determine whether to display a dialog box before printing, in which the user can set several spool parameters for print formatting. Possible values: ' ' display no print parameter screen 'X' display print parameter screen Default value: 'X'

OPTIONS	Use parameter OPTIONS to set several options for print formatting. The parameter has the structure ITCPO. The user can change some of the defined settings on the print control screen. Structure: ITCPO Default value: SPACE
APPLICATION	For the device type SCREEN, the system displays the text formatting on the screen. This requires an interface in which the different menu entries are defined. The same applies if the user chooses to display a print view on the screen for the other device types. Enter one of the interface names provided by SAPscript. You usually use the interface that is assigned to the respective text object in table TTXOB. Reference field: TTXOB-TDAPPL Default value: 'TX'
ARCHIVE_INDEX	Enter the index information for the print output you want to archive. This information (DARA line) is stored in the archive together with the print output. Thus, you can use the index information to access the print output directly in the archive. Structure: TOA_DARA Default value: SPACE
ARCHIVE_PARAMS	The system interprets the settings passed in this parameter when archiving the output. The archive parameters have the ABAP Dictionary structure ARC_PARAMS. Structure: ARC_PARAMS Default value: SPACE

Import parameters:

LANGUAGE	The parameter tells you which language variant of the form the system actually used. Reference field: THEAD-TDSPRAS
RESULT	The parameter contains results of the print formatting process. By comparing the corresponding fields of parameter OPTIONS with those of parameter RESULT, you can determine whether the user made changes to any settings on the print control screen. Structure: ITCPP

NEW_ARCHIVE_PARAMS	The parameter contains results of the archiving process, including the archive parameters the user changed on the print control screen. The parameter has the ABAP Dictionary structure ARC_PARAMS. Structure: ARC_PARAMS

Exceptions:

CANCELED	When starting SAPscript print formatting, the system displayed a selection screen, on which the user can enter settings for the output, such as: • Printer name • Information on the cover page • Page selection • Number of copies The user did not call any subsequent actions allowed on this screen, but canceled the output formatting instead. The function module was ended without further action. No form is open for output anymore.
DEVICE	The parameter DEVICE contains an invalid device type.
FORM	The parameter FORM contains the name of a form that the system could not find. Possible reasons: • The form does not exist. • There is no active version of this form. SAPscript first searches for the form in the current client and in the specified language. If the form does not exist there, it tries the original language of the form. If the form is still not found, it searches in client 0, first in the specified language, than in the original form language.
OPTIONS	The parameter OPTIONS contains invalid values for the formatting options. Possible errors: • The output device specified in field TDDEST does not exist. • The field TDPAGESLCT for selecting the pages to be printed contains invalid characters.
UNCLOSED	The system was told to open a new form even though an old form is still active. The old form must be closed first (CLOSE_FORM or END_FORM).

CLOSE_FORM

The function module closes the form opened using OPEN_FORM. The system executes any terminating processing steps for the last opened form.

You must use this function module to close form printing. Otherwise, no output appears on printer or screen.

Function call:

```
CALL FUNCTION  'CLOSE_FORM'
     IMPORTING  RESULT   =
     TABLES     OTFDATA  = ?...
     EXCEPTIONS UNOPENED =
```

Import parameters:

RESULT	The parameter contains results of the print formatting process. By comparing the corresponding fields of parameter OPTIONS with those of parameter RESULT, you can determine whether the user made changes to any settings on the print control screen. Structure: ITCPP Among others, the structure ITCPP contains a field with the name of USEREXIT. This field tells you how the user left the print view: Characters E, B, or C: EXIT <-> E BACK <-> B CANCEL <-> C

Table parameters:

OTFDATA	If in the parameter OPTIONS the field TDGETOTF contains 'X', the system returns the formatted output in the OTF format in the optional table parameter OTFDATA. In this case, the system does not output anything to printer, screen or fax/telex/teletex. Structure: ITCOO

Exceptions:

UNOPENED	The system could not execute the current form function, since the form output was not yet initialized using OPEN_FORM.

START_FORM

In-between the function modules OPEN_FORM and CLOSE_FORM, you can use different forms. This allows you to combine several different forms into one print output. However, you can combine only those forms that have the same page format.

To switch forms, use the function module START_FORM. If another form is still open, you must close it first using END_FORM.

If you specify no form name when calling START_FORM, the system restarts the last open form. If after OPEN_FORM no form was activated yet, the system leaves the function module with the exception UNUSED.

Function call:

```
CALL FUNCTION  'START_FORM'
     EXPORTING  FORM          = SPACE
                LANGUAGE      = SPACE
                STARTPAGE     = SPACE
                PROGRAM       = SPACE
                ARCHIVE_INDEX = SPACE
     IMPORTING  LANGUAGE      =
     EXCEPTIONS FORM          =
                FORMAT        =
                UNENDED       =
                UNOPENED      =
                UNUSED        =
```

Export parameters:

FORM	The parameter contains the name of the form you want to use for printing. If you specify no form here, the system restarts the last active form. Default value: SPACE
LANGUAGE	Forms are language-dependent. Enter the desired language here. If the form does not exist in this language, the system tries to call the form in its original language. If the parameter LANGUAGE is empty, the system uses the language of the last active form. Reference field: THEAD-TDSPRAS Default value: SY-LANGU

STARTPAGE	Usually, SAPscript starts with the page specified as start page in the form definition. If you want to start output with another form page, enter the name of the desired form page here. If the desired page is not defined, the system uses the start page defined in the form. Default value: SPACE
PROGRAM	To replace program symbols, SAPscript must know which active program contains the work areas for the values to be passed. If you omit the parameter, the system searches for the field values in the program that was specified in the parameter OPTIONS (field TDPROGRAM) of OPEN_FORM. If you enter a program name, the system replaces the program symbols with the values from this program up to the next END_FORM. Reference field: SY-REPID Default value: SPACE
ARCHIVE_INDEX	Enter index information for the print output you want to archive. The system stores this information (DARA line) together with the print output in the archive. You can then access this special print output in the archive using the index information. Structure: TOA_DARA Default value: SPACE

Import parameters:

LANGUAGE	The parameter tells you which language variant of the form the system actually used. Reference field: THEAD-TDSPRAS

Exceptions:

FORM	The parameter FORM contains the name of a form which could not be found. Possible reasons: • The form does not exist. • There is no active version of this form. SAPscript first searches for the form in the current client and in the specified language. If the form does not exist there, it tries the original language of the form. If the form is still not found, it searches in client 0, first in the specified language, than in the original form language.

FORMAT	Within a print output defined between OPEN_FORM and CLOSE_FORM, you must use forms of the same page format. The page format of the current form differs from that of the previously called forms. Include only those forms into one spool request that have the same format (for example, DINA4). Page orientation is of no consequence; that means, you can mix pages in landscape and portrait format.
UNENDED	The last form opened is still open. It must first be closed using END_FORM or the form output must be closed using CLOSE_FORM.
UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.
UNUSED	One of the parameters FORM or LANGUAGE contains only blanks and no form has been opened yet whose name or language could be used as defaults.

WRITE_FORM

The system outputs the form element specified in parameter ELEMENT into the currently opened form.

In the parameter WINDOW you can specify the name of a window for the output. Remember that the form element must be defined in this window. The parameter FUNCTION specifies how to merge the text lines to be output with any existing contents in the window. In this case, there are differences between the different window types or areas.

Function call:

```
CALL FUNCTION  'WRITE_FORM'
     EXPORTING  ELEMENT       = SPACE
                WINDOW        = 'MAIN'
                FUNCTION      = 'SET'
                TYPE          = 'BODY'
     IMPORTING  PENDING_LINES =
     EXCEPTIONS ELEMENT       =
                FUNCTION      =
                TYPE          =
                UNOPENED      =
                UNSTARTED     =
                WINDOW        =
```

Export parameters:

ELEMENT	Specify the name of the text element you want to output into the form window specified in the parameter WINDOW. The element must be defined in that form window. If you specify no element, the system uses the default element, if one is defined in the form. Default value: SPACE
WINDOW	Specify the name of the window into which you want to output the form element specified in the parameter ELEMENT. Default value: 'MAIN'

FUNCTION	The parameter determines how to output the text element into the respective window. The output type depends on the window type and area: Window type MAIN, area BODY: 'SET' append to previous output 'APPEND' same as SET 'DELETE' no effect Window type MAIN, areas TOP and BOTTOM; all other windows: 'SET' delete old window or area contents and output the element 'APPEND' append the element to the existing elements 'DELETE' delete the specified element from the window or area ⚠ DELETE in the TOP area (headings) takes effect only on the next page. You can no longer delete any heading from the TOP area after outputting text to the BODY area. Default value: 'SET'
TYPE	The system interprets this parameter only for output to the main window. The parameter determines the area of the main window into which you want to output the element. Possible values: 'TOP' header area 'BODY' main area 'BOTTOM' footer area Default value: 'BODY'

Import parameters:

PENDING_LINES	If a text is output to the BOTTOM area of the main window (TYPE = 'BOTTOM'), there may be not enough space left on the current output page. The system then internally flags this text for output into the BOTTOM area of the next page. The actual output is pending. If output is pending, the parameter PENDING_LINES contains 'X', and the print program can react accordingly. For example, an explicit page break at the text end (NEW-PAGE) could implicitly trigger the pending BOTTOM output on the next page.

Exceptions:

ELEMENT	The parameter ELEMENT contains the name of a form element that the system could not find. Possible reasons: • The element does not exist. An element refers to a window, and the specified window does not contain the element. If no window is specified, the system searches for the element in the main window. • The specified element is not defined in the form. • The form version that contains the text element in the specified form window is not active.
FUNCTION	The function specified in parameter FUNCTION is unknown. For this parameter, the following values are allowed: SET APPEND DELETE
TYPE	The type of window area specified in parameter TYPE is invalid. Depending on the window type, these entries are valid: • BODY: for all windows • TOP: only for the main window • BOTTOM: only for the main window
UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.

UNSTARTED	No form was opened yet. Possible reasons: • The form processing was started using OPEN_FORM without specifying a form name, but no form was opened yet using START_FORM. • The last used form was closed using END_FORM, but no new form was opened using START_FORM. • The last filled page of the current form has no subsequent page. In this case, the system automatically terminates form printing after this page. You need no explicit END_FORM call. • In the current form, no page contains a main window, but a text element shall be output in the main window.
WINDOW	The form window specified in parameter WINDOW does not exist in the current form. Possible reasons: • A wrong window name was specified. • The form version, which contains the specified window, is not active.

WRITE_FORM_LINES

The function module outputs the text lines in table LINES into the specified form window. The text lines must have the SAPscript ITF format. From the data in the text header, the system uses only the field TDSTYLE to apply the formatting attributes defined in the specified style for this text. If the field is empty, the system uses the identically named formatting attributes (character and paragraph formats) of the form.

Use parameter WINDOW to specify into which of the windows defined in the form you want to output the text. You can specify any window used in the form. The parameter FUNCTION determines how to merge the text lines to be output with any existing contents in the window. There are differences between the different window types or areas.

Function call:

```
CALL FUNCTION  'WRITE_FORM_LINES'
     EXPORTING  HEADER        = ?...
                WINDOW        = 'MAIN'
                FUNCTION      = 'SET'
                TYPE          = 'BODY'
     IMPORTING  PENDING_LINES =
                FROMPAGE      =
     TABLES     LINES         = ?...
     EXCEPTIONS FUNCTION      =
                TYPE          =
                UNOPENED      =
                UNSTARTED     =
                WINDOW        =
```

Export parameters:

HEADER	The parameter contains the header of the text module you want to output in the current form. For the formatting process, the system uses only the entries in the header fields TDSTYLE and TDFORM. Structure: THEAD
WINDOW	Enter the name of the window into which you want to output the form element specified in parameter ELEMENT. Default value: 'MAIN'

FUNCTION	The parameter determines how to output the text element into the respective window. The output type depends on the window type and area: Window type MAIN, area BODY: 'SET' append to previous output 'APPEND' same as SET 'DELETE' no effect Window type MAIN, areas TOP and BOTTOM; all other windows: 'SET' delete old window or area contents and output the element 'APPEND' append the element to the existing elements 'DELETE' no effect Default value: 'SET'
TYPE	The system interprets this parameter only for output to the main window. The parameter determines the area of the main window into which you want to output the element. Possible values: 'TOP' header area 'BODY' main area 'BOTTOM' footer area Default value: 'BODY'

Import parameters:

PENDING_LINES	If a text is output to the BOTTOM area of the main window (TYPE = 'BOTTOM'), there may be not enough space left on the current output page. The system then internally flags this text for output into the BOTTOM area of the next page. The actual output is pending. If output is pending, the parameter PENDING_LINES contains 'X', and the print program can react accordingly. For example, an explicit page break at the text end (NEW-PAGE) could implicitly trigger the pending BOTTOM output on the next page.
FROMPAGE	Use FROMPAGE to determine on which form page the output of the text starts.

Table parameters:

WRITE_FORM_LINES

LINES	Enter the name of the table that contains the text lines you want to print. Structure: TLINE

Exceptions:

FUNCTION	The function specified in parameter FUNCTION is unknown. For this parameter, the following values are allowed: SET APPEND DELETE
TYPE	The type of window area specified in parameter TYPE is invalid. Depending on the window type, these entries are valid: • BODY: for all windows • TOP: only for the main window • BOTTOM: only for the main window
UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.
UNSTARTED	No form was opened yet. Possible reasons: • The form processing was started using OPEN_FORM without specifying a form name, but no form was opened yet using START_FORM. • The last used form was closed using END_FORM, but no new form was opened using START_FORM. • The last filled page of the current form has no subsequent page. In this case, the system automatically terminates form printing after this page. You need no explicit END_FORM call. • In the current form, no page contains a main window, but a text element shall be output in the main window.
WINDOW	The form window specified in parameter WINDOW does not exist in the current form. Possible reasons: • A wrong window name was specified. • The form version, which contains the specified window, is not active.

END_FORM

END_FORM ends the currently open form and executes the required termination processing. After calling this function module, no more form is active. For further output, you must start a new form using START_FORM.

⚠

END_FORM does not replace CLOSE_FORM, that is, you must always close any SAPscript output using CLOSE_FORM.

Function call:

```
CALL FUNCTION  'END_FORM'
     IMPORTING  RESULT   =
     EXCEPTIONS UNOPENED =
```

Import parameters:

RESULT	The parameter contains results of the print formatting process. By comparing the corresponding fields of parameter OPTIONS with those of parameter RESULT, you can determine whether the user made changes to any settings on the print control screen. Structure: ITCPP

Exceptions:

UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.

CONTROL_FORM

Use CONTROL_FORM to pass SAPscript control statements to the form.

Function call:

```
CALL FUNCTION  'CONTROL_FORM'
     EXPORTING  COMMAND    = ?...
     EXCEPTIONS UNOPENED   =
                UNSTARTED  =
```

Export parameters:

COMMAND	Enter the SAPscript statement you want to execute in ITF format, but without the statement paragraph attribute '/:'.

Exceptions:

UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.
UNSTARTED	No form was opened yet. Possible reasons: • The form processing was started using OPEN_FORM without specifying a form name, but no form was opened yet using START_FORM. • The last used form was closed using END_FORM, but no new form was opened using START_FORM. • The last filled page of the current form has no subsequent page. In this case, the system automatically terminates form printing after this page. You need no explicit END_FORM call. • In the current form, no page contains a main window, but a text element shall be output in the main window.

READ_FORM_ELEMENTS

The function module fills a table with all text elements that appear in one form.

If you specify no form name, the system includes all elements of the currently open form. If you specify a form, the system uses the information about the active version of the form, retrieved from the database.

Function call:

```
CALL FUNCTION  'READ_FORM_ELEMENTS'
     EXPORTING  FORM     = SPACE
                LANGUAGE = SPACE
     TABLES     ELEMENTS = ?...
     EXCEPTIONS FORM     =
                UNOPENED =
```

Export parameters:

FORM	Specify the name of the form whose element list you want to create. If you leave the field blank, the system uses the currently active form. Default value: SPACE
LANGUAGE	Specify the desired form language. Default value: SPACE

Table parameters:

ELEMENTS	The table contains all windows defined in a form together with the corresponding text elements. For each text element, the system specifies the number of text lines (editor lines) it consists of. Structure: ITCWE

Exceptions:

FORM	The parameter FORM contains the name of a form which could not be found. Possible reasons: • The form does not exist. • There is no active version of this form. SAPscript first searches for the form in the current client and in the specified language. If the form does not exist there, it tries the original language of the form. If the form is still not found, it searches in client 0, first in the specified language, than in the original form language.
UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.

READ_FORM_LINES

Use this function module to transfer the lines of a form element into an internal table.

If you specify no form name, the system transfers the text lines of the currently open form. If you specify a form, the system uses the text lines of the active version of the form from the database.

Function call:

```
CALL FUNCTION  'READ_FORM_LINES'
     EXPORTING  FORM     = SPACE
                LANGUAGE = SPACE
                WINDOW   = 'MAIN'
                ELEMENT  = SPACE
     TABLES     LINES    = ?...
     EXCEPTIONS ELEMENT  =
                FORM     =
                UNOPENED =
```

Export parameters:

FORM	Specify the name of the form whose element list you want to create. If you leave the field blank, the system uses the currently active form. Default value: SPACE
LANGUAGE	Specify the desired form language. Default value: SPACE
WINDOW	Enter the name of the window in which to find the form element. If you leave the field blank, the system searches for the form element in the main window. Default value: 'MAIN'
ELEMENT	Enter the name of the form element whose text lines you want to read. If you leave the field blank, the system searches for the 'nameless' element. Default value: SPACE

Table parameters:

LINES	The table receives the lines of the specified element. Structure: TLINE

Exceptions:

ELEMENT	The parameter ELEMENT contains the name of a form element which the system could not find. Possible reasons: • The element does not exist. An element refers to a window, and the specified window does not contain the element. If no window is specified, the system searches for the element in the main window. • The specified element is not defined in the form. • The form version that contains the text element in the specified form window is not active.
FORM	The parameter FORM contains the name of a form which could not be found. Possible reasons: • The form does not exist. • There is no active version of this form. SAPscript first searches for the form in the current client and in the specified language. If the form does not exist there, it tries the original language of the form. If the form is still not found, it searches in client 0, first in the specified language, than in the original form language.
UNOPENED	The current form function could not be executed, since the form output was no yet initialized using OPEN_FORM.

CONVERT_TEXT

The function module **CONVERT_TEXT** provides the following format conversions:

ITF text in table **ITF_LINES**	→	RTF text in table **FOREIGN**
ITF text in table **ITF_LINES**	→	ASCII text in table **FOREIGN**
RTF text in table **FOREIGN**	→	ITF text in table **ITF_LINES**
ASCII text in table **FOREIGN**	→	ITF text in table **ITF_LINES**

For conversions within the SAPscript format ITF, use the function module **EXCHANGE_ITF**.

You must enter the following specifications:

DIRECTION = 'EXPORT' FORMAT_TYPE = 'RTF'

Since the system prefixes the result table with information on author, creation date, etc., you must fill the parameter **HEADER, especially the fields specifying style and form of the text** (HEADER-TDSTYLE and HEADER-TDFORM).

In **FORMATWIDTH, you can enter the line width of the result text**.

In **CODEPAGE, enter the character set numbers from the spool ad**ministration. For RTF conversion, you can select the character sets **1103**, **1110,** or **1133. The system ignores any other entries and uses the default character set** 1133 instead. If you want to read the RTF text into WinWord, we recommend to use character set 1133.

TABLETYPE contains the type of table FOREIGN. If you want to compress the table, that is, give it a file-like structure, use the entire width of 134 characters, and separate the individual logical lines with carriage return and line feed, you must use TABLETYPE = 'BIN'. If you use TABLETYPE = 'ASC', you get a normal lines table.

WORD_LANGU has no effect; **SSHEET** and **WITH_TAB** are explained at the 'Format conversion'.

DIRECTION = 'EXPORT' FORMAT_TYPE = 'ASCII'

All ITF information is lost with this conversion; the only formats are Newline and Tab. By choosing the line width FORMATWIDTH accordingly, you can achieve a certain page formatting. You can convert the ITF text into an ASCII text of any character set **CODEPAGE** (number form spool administration).

TABLETYPE contains the type of table FOREIGN. If you want to compress the table, that is, give it a file-like structure, use the entire width of 134 characters, and separate the individual logical lines with carriage return and line feed, you must use TABLETYPE = 'BIN'. If you use TABLETYPE = 'ASC', you get a normal lines table.

In short, you must specify DIRECTION and FORMAT_TYPE; the only import parameters for which entries are useful, are CODEPAGE, FORMATWIDTH, and TABLETYPE.

DIRECTION = 'IMPORT' FORMAT_TYPE = 'RTF'

For WORD_LANGU enter the language of the MS Word version you use to allow the system to interpret language-dependent elements in the RTF text. The default is the R/3 system language.

TABLETYPE contains the type of table FOREIGN. If you created this table, for example, by uploading a local file in binary format, which means that all lines except the last are filled completely and the lines may contain carriage return and line feed, you must use TABLETYPE = 'BIN'. If the table is a normal lines table, use TABLETYPE = 'ASC'.

The parameters **FORMATWIDTH** and **CODEPAGE** have no effect; **SSHEET** and **WITH_TAB** are explained at the 'Format conversion'.

HEADER is used only in connection with a print format conversion (WITH_TAB = 'X'). In this case, the system fills the parameter NEWHEADER with the new current values.

DIRECTION = 'IMPORT' FORMAT_TYPE = 'ASCII'

The table containing the ASCII text is converted line by line. In the ITF target text, the system separates the individual ITF lines with NEWLINE. The ASCII tab is converted to the ITF tab (',,').

In **CODEPAGE, you must enter the character set of the source text** (that is, its number in the spool administration); the system then converts the characters to the system character set. If you leave CODEPAGE blank, no character conversion occurs.

TABLETYPE contains the type of table FOREIGN. If you created this table, for example, by uploading a local file in binary format, which means that all lines except the last are filled completely and the lines may contain carriage return and line feed, you must use TABLETYPE = 'BIN'. If the table is a normal lines table, use TABLETYPE = 'ASC'.

The system ignores the parameters **FORMATWIDTH, SSHEET, WITH_TAB, WORD_LANGU, and HEADER**.

Format conversion

If you set the parameter WITH_TAB = 'X', the system triggers a format conversion for FORMAT_TYPE = 'ITF' and FORMAT_TYPE = 'RTF'. Beforehand, you must use transaction SE74 (Format conversion) to specify which character and paragraph names or Word templates to map on other character and paragraph names or Word templates.

The system converts for

ITF → RTF:

HEADER-TDSTYLE or HEADER-TDFORM into SSHEET

RTF → ITF:

SSHEET into HEADER-TDSTYLE or HEADER_TDFORM

ITF → ITF (in both directions)

In this case, the parameter SSHEET **must** contain 'S' for style or 'F' for form as prefix.

To an ITF text, the style 'STYLE1' and the form 'FORM1' are allocated. You want to format it using the form 'FORM2'. In this case:

HEADER-TDSTYLE = 'STYLE1', HEADER-TDFORM = 'FORM1' and SSHEET = 'FFORM2'.

Function call:

```
CALL FUNCTION  'CONVERT_TEXT'
     EXPORTING  CODEPAGE    = SPACE
                DIRECTION   = 'EXPORT'
                FORMATWIDTH = 72
                FORMAT_TYPE = 'RTF'
                HEADER      = SPACE
                SSHEET      = SPACE
                WITH_TAB    = SPACE
                WORD_LANGU  = SY-LANGU
                TABLETYPE   = 'ASC'
     IMPORTING  NEWHEADER   =
     TABLES     FOREIGN     = ?...
                ITF_LINES   = ?...
```

Export parameters:

CODEPAGE	Character set number from the spool administration. If DIRECTION = ‘EXPORT’, CODEPAGE contains the target character set, if DIRECTION = ‘IMPORT’ the source character set. Reference field: TCP02-CPCODEPAGE Default value: SPACE
DIRECTION	‘IMPORT’ or ‘EXPORT’
FORMATWIDTH	Default value: 72
FORMAT_TYPE	The parameter can have the values of the following text formats: RTF (RichTextFormat) ASCII Default value: 'RTF'
HEADER	Text header of the source text. The text header contains the description of a text module, such as short text, creator, and so on. Structure: THEAD Default value: SPACE

SSHEET	Name of a template (*.dfv or *.dot). To trigger a format conversion, you must enter the symbolic name for which you maintained a conversion to the respective SAPscript style or form in transaction SE74. In addition, you must set the parameter WITH_TAB to 'X'. Default value: SPACE
WITH_TAB	If WITH_TAB = 'X' and the parameter SSHEET is filled, the system converts character and paragraph formats. Source and target formats can be a SAP*script* style, a SAP*script* form, or an MS Word template. Using transaction SE74 (Format conversion), you must specify which character or paragraph names or Word templates match. Default value: SPACE
WORD_LANGU	Only for RTF conversion: language of the word processing program that created the RTF file. Default value: SY-LANGU
TABLETYPE	DIRECTION = 'IMPORT' **TABLETYPE** contains the type of table FOREIGN. If you created this table, for example, by uploading a local file in binary format, which means that all lines except the last are filled completely and the lines may contain carriage return and line feed, you must use TABLETYPE = 'BIN'. If the table is a normal lines table, use TABLETYPE = 'ASC'. DIRECTION = 'EXPORT' **TABLETYPE** contains the type of table FOREIGN. If you want to compress the table, that is, give it a file-like structure, use the entire width of 134 characters, and separate the individual logical lines with carriage return and line feed, you must use TABLETYPE = 'BIN'. If you use TABLETYPE = 'ASC', you get a normal lines table. Default value: 'ASC'

Import parameters:

NEWHADER	Text header of the SAPscript result text Structure: THEAD

Table parameters:

FOREIGN	Foreign text table Structure: TLINE

CONVERT_TEXT

ITF_LINES	SAPscript ITF table Structure: TLINE
LINKS_TO_CONVERT	Not implemented!

EXCHANGE_ITF

The function module **EXCHANGE_ITF** is interactive. You use it to convert the character and paragraph names of a style or form into those of another style or form. You specify the desired new style or form allocation in a dialog box. To specify which original names match which new names, use transaction SE74 (Format conversion), column "Format conversion in SAPscript texts".

Function call:

```
CALL FUNCTION  'EXCHANGE_ITF'
     EXPORTING  FORMATWIDTH = 72
                HEADER      = ?...
     IMPORTING  NEWHEADER   =
     TABLES     ITF_LINES   = ?...
     EXCEPTIONS CANCELED   =
```

Export parameters:

FORMATWIDTH	Maximum line width of the result text. If table TTXOB contains an entry for the current text object, the system overwrites the parameter FORMATWIDTH with this value. Statement lines are never split. Default value: 72
HEADER	Text header of the source text. Structure: THEAD

Import parameters:

NEWHEADER	Text header of the result text. Structure: THEAD

Table parameters:

ITF_LINES	The system reads the text to be converted from table ITF_LINES, processes it and returns it in the same table. Structure: TLINE

Exceptions:

CANCELED	Conversion canceled by user

IMPORT_TEXT

The function module **IMPORT_TEXT** uploads a local file into the R/3 system and then executes one of the following conversions, depending on the file format FORMAT_TYPE:

- Local ITF file **FILE** → ITF text in table **ITF_LINES**
- Local RTF file **FILE** → ITF text in table **ITF_LINES**
- Local ASCII file **FILE** → ITF text in table **ITF_LINES**

You must enter the following specifications:

FORMAT_TYPE = 'ITF'

Here, you can use ITF files with header information (for more information on this file type, see documentation of the report RSTXLITF) as well as ITF texts without a preface. If header information exists, it is written to the export parameter **NEWHEADER and the system formats the text: The line width is set to the value defined in table** TTXOB for the current text object. The parameters **HEADER**, **FORMATWIDTH**, **CODEPAGE**, **WORD_LANGU**, **SSHEET,** and **WITH_TAB** have no effect.

FORMAT_TYPE = 'RTF'

In **WORD_LANGU, specify the language of the MS Word version used to enable the system to interpret language-dependent elements in the RTF file**. The default is the R/3 system language. The parameters **FORMATWIDTH** and **CODEPAGE** have no effect, for a description of **SSHEET** and **WITH_TAB, see "Format conversion" below. HEADER** is used only in connection with a format conversion (WITH_TAB = 'X'), when the system fills the parameter NEWHEADER with new, current values.

FORMAT_TYPE = 'ASCII'

The system converts carriage return (x0D) and line feed (x0A) into newline ('/ ') and matches form feed (x0c) with the statement '/: NEW-PAGE' and the ASCII tab with the ITF tab (',,').

In **CODEPAGE, you must specify the character set of the source file (that is, its number in the s**pool administration); the system then converts the characters into the system character set. If you specify no value in CODEPAGE, no character set conversion occurs.
The system does not evaluate the parameters **FORMATWIDTH**, **SSHEET**, **WITH_TAB**, **WORD_LANGU,** and **HEADER**.

Format conversion

If the parameter WITH_TAB = 'X', the system starts a format conversion, provided that FORMAT_TYPE = 'RTF'. Beforehand, you must use transaction SE74 (Format conversion) to specify which Word templates to match with which character and paragraph formats of different names. The system converts

		HEADER-TDSTYLE
SS	i	or
HE	n	HEADER-TDFORM
ET		

Function call:

```
CALL FUNCTION  'IMPORT_TEXT'
     EXPORTING  CODEPAGE        = ?...
                FILE            = ?...
                FORMAT_TYPE     = 'ITF'
                HEADER          = SPACE
                SSHEET          = SPACE
                WITH_TAB        = SPACE
                WORD_LANGU      = SY-LANGU
     IMPORTING  NEWHEADER       =
     TABLES     ITF_LINES       = ?...
     EXCEPTIONS FILE_OPEN_ERROR =
                FILE_READ_ERROR =
                UPLOAD_ERROR    =
```

Export parameters:

CODEPAGE	Character set of the source file. You must enter the character set numbers from the spool administration. Reference field: TCP02-CPCODEPAGE
FILE	Name of the file to be uploaded. If this file does not exist or is unreadable, the system triggers the respective exceptions. ➡ Make sure to use the correct case for the file name. Fully specify the file directory. Reference field: RLGRAP-FILENAME
FORMAT_TYPE	File format ('ITF', 'RTF', or 'ASCII') Default value: 'ITF'
HEADER	Structure: THEAD Default value: SPACE

SSHEET	Name of a template (*.dfv or *.dot). To trigger a format conversion, you must enter the symbolic name for which you maintained a conversion to the respective SAPscript style or form in transaction SE74. In addition, you must set the parameter WITH_TAB to 'X'. Default value: SPACE
WITH_TAB	If WITH_TAB = 'X' and the parameter SSHEET is filled, the system converts character and paragraph formats. Source and target formats can be a SAP*script* style, a SAP*script* form, or an MS Word template. Using transaction SE74 (Format conversion), you must specify which character or paragraph names or Word templates match. Default value: SPACE
WORD_LANGU	Only for RTF conversion: language of the word processing program that created the RTF file. Default value: SY-LANGU

Import parameters:

NEWHEADER	Text header of the SAPscript result text. The text header contains a description of a text module, such as short text, creator, and so on. The structure is determined in table THEAD. Structure: THEAD

Table parameters:

ITF_LINES	Text table of the result text. Structure: TLINE

Exceptions:

FILE_OPEN_ERROR	File cannot be opened.
FILE_READ_ERROR	File cannot be read (completely). Possible reason: • Read error of the operating system (no SAP error) or upload incomplete due to incorrect GUI installation (work directory write-protected).
UPLOAD_ERROR	Other errors when uploading the file.

EXPORT_TEXT

The function module **EXPORT_TEXT** converts an ITF text into the desired format FORMAT_TYPE and then saves it in a local file:

- ITF text in table **ITF_LINES** → local ITF file **FILE**
- ITF text in table **ITF_LINES** → local RTF file **FILE**
- ITF text in table **ITF_LINES** → local ASCII file **FILE**

When exporting the formats ASCII and RTF, the system resolves text includes and control structures (/: IF, /: ELSE, /: ENDIF, /: CASE, /: WHEN, /: ENDCASE) and replaces text, standard, and system symbols.

You must enter the following specifications:

FORMAT_TYPE = 'ITF'

Since the result file is provided with text header information (for more details on this file type, see documentation of report RSTXSITF), you should fill the parameter **HEADER**.

The parameters **FORMATWIDTH**, **CODEPAGE**, **SSHEET,** and **WITH_TAB** have no effect.

FORMAT_TYPE = 'RTF'

Since the result file is provided with information on author, creation date, and so on, you should fill the HEADER parameter. In any case, you must specify style and form of the text (HEADER-TDSTYLE and HEADER-TDFORM).

In **FORMATWIDTH, specify the line width of the result text. In CODEPAGE, you must enter the characters set numbers from the s**pool administration. For RTF conversion, you can choose between the character sets **1103**, **1110,** or **1133. The system ignores all other assignments and uses the character set 1133 instead**. If you want to read the created RTF text using WinWord, we recommend the character set 1133. **SSHEET** and **WITH_TAB** are explained in 'Format conversion' below.

FORMAT_TYPE = 'ASCII'

All ITF information is lost with this conversion; the only formats are newline and tab. By choosing the line width FORMATWIDTH accordingly, you can achieve a certain page formatting. You can convert the ITF text into an ASCII text of any character set **CODEPAGE** (number form spool administration).

In short, you must specify FORMAT_TYPE; the only import parameters for which entries are useful, are CODEPAGE and FORMATWIDTH.

Format conversion

If you set the parameter WITH_TAB = 'X', the system triggers a format conversion for FORMAT_TYPE = 'RTF'. Beforehand, you must use transaction SE74 (Format conversion) to specify which character and paragraph names or Word templates to match which other character and paragraph names or Word templates.

The system converts

HEADER-TDSTYLE		
or	in	SSHEET

EXPORT_TEXT

HEADER-TDFORM		

Function call:

```
CALL FUNCTION    'EXPORT_TEXT'
     EXPORTING   CODEPAGE          = SPACE
                 FILE              = ?...
                 FORMATWIDTH       = 72
                 FORMAT_TYPE       = 'RTF'
                 HEADER            = SPACE
                 SSHEET            = SPACE
                 WITH_TAB          = SPACE
                 TAB_SUBSTITUTE    = 'X09  '
     TABLES      ITF_LINES         = ?...
     EXCEPTIONS  DOWNLOAD_ERROR    =
                 FILE_OPEN_ERROR   =
                 FILE_WRITE_ERROR  =
```

Export parameters:

CODEPAGE	Specify the character set numbers from the spool administration. Reference field: TCP02-CPCODEPAGE Default value: SPACE
FILE	Name of the file you want to create on the presentation server (if necessary, with leading directory name). If the directory does not exist or if the system cannot open the file for writing, it triggers the respective exceptions. Reference field: RLGRAP-FILENAME
FORMATWIDTH	Line width of the target file. Default value: 72
FORMAT_TYPE	Format of the target file ('ITF', 'RTF', or 'ASCII'). Default value: 'RTF'

HEADER	Text header of the source text. The text header contains a description of a text module, such as short text, creator, and so on. Structure: THEAD
SSHEET	Name of a template (*.dfv or *.dot). To trigger a format conversion, you must enter the symbolic name for which you maintained a conversion to the respective SAPscript style or form in transaction SE74. In addition, you must set the parameter WITH_TAB to 'X'. Default value: SPACE
WITH_TAB	If WITH_TAB = 'X' and the parameter SSHEET is filled, the system converts character and paragraph formats. Source and target formats can be a SAP*script* style, a SAP*script* form, or an MS Word template. Using transaction SE74 (Format conversion), you must specify which character or paragraph names or Word templates match. Default value: SPACE
TAB_SUBSTITUTE	Substitution value for the SAPscript tab. The parameter is used only for a conversion from ITF to ASCII. You define the substitution value in a character field of length 5. The first character defines the type of substitution: 'C' substitutes the tab with a character string of up to four characters. TAB_SUBSTITUTE = 'C<<>>' "before tab,,after tab " in SAPscript becomes "before tab <<>>after tab " in the ASCII file. 'X' substitutes the tab with one or two binary characters. TAB_SUBSTITUTE = 'X09_ _'. The SAPscript tab ",," becomes the hexadecimal value 09. ‘_' A blank substitutes the tab with 1 to 99 blanks. TAB_SUBSTITUTE = '_ 8_ _ _'. "before tab,,after tab " in SAPscript becomes "before tab_ _ _ _ _ _ _ _after tab " in the ASCII file. Default value: 'X09_ _'

Table parameters:

ITF_LINES	SAPscript text table you want to export. Structure: TLINE

Exceptions:

FILE_OPEN_ERROR	File cannot be opened.
FILE_WRITE_ERROR	File cannot be written.
DOWNLOAD_ERROR	Other errors when downloading file.

TRANSFER_TEXT

The function module **TRANSFER_TEXT** is interactive. It calls either the function module IMPORT_TEXT with the parameter ITF_LINES = TABLE_IMP or the function module EXPORT_TEXT with the parameter ITF_LINES = TABLE_EXP.

The parameter **CHFORMAT** determines whether you can select style and form for the result text for ITF and RTF imports. If such a selection is not possible *(*CHFORMAT = ' '*),* the system uses HEADER-TDSTYLE and HEADER-TDFORM.

Function call:

```
CALL FUNCTION  'TRANSFER_TEXT'
     EXPORTING  CHFORMAT  = 'X'
                HEADER    = SPACE
                DIRECTION = SPACE
     IMPORTING  NEWHEADER =
     TABLES     TABLE_EXP = ?...
                TABLE_IMP = ?...
     EXCEPTIONS CANCELED =
```

Export parameters:

CHFORMAT	The parameter determines whether you are allowed to change the style and form allocation. Possible values: ' ' changes allowed 'X' changes not allowed Default value: 'X'
HEADER	Text header of the source text. Structure: THEAD Default value: SPACE
DIRECTION	'IMPORT', 'EXPORT', or SPACE. For DIRECTION = SPACE, you can choose in a dialog box between file import (from local file) and file export (to local file) and between different text formats. If you specify ‘IMPORT’ or ‘EXPORT’, you can determine only the text format. Default value: SPACE

Import parameters:

NEWHEADER	Text header of the result text. Structure: THEAD

Table parameters:

TABLE_EXP	Table to be exported. Structure: TLINE
TABLE_IMP	Table containing import result. Structure: TLINE

Exceptions:

CANCELED	Transfer canceled by user.

CONVERT_TEXT_R2

The function module converts texts from SAP R/2 text format to SAPscript format of the R/3 system.

Apart form converting the format structure of the text lines, the function module considers the following attributes of R/2 texts:

- line attributes
- control statements
- symbols

During the conversion, several incompatibilities can occur. These incompatibilities are grouped into levels. The parameter ERRORLEVEL returns the level number of the most serious incompatibility.

Line format

The line format of R/2 texts is similar to that of R/3 texts. Both lines consist of a field that contains the line format and a field with the actual line contents. Conversion of line formats is described under 'Line attributes'.

A text line in R/2 can consist of up to 220 characters; in SA*P*script, the line contents is limited to 132 characters. If an R/2 text line consists of more than 132 characters, the system transfers the remainder into a subsequent line with no paragraph format.

Line attributes

SAPscript does not distinguish between fixed and variable lines, but knows only what R/2 calls variable lines. To convert the R/2 line formats, the following rules apply:

- At lines with the attributes A and F always a new SAPscript paragraph starts.
- Adjacent lines with the attribute V are considered as one paragraph. If before the first V line a line with attribute F appears, then the first V line is the beginning of a new SAPscript paragraph. If the line before the V block has the attribute A, the V lines are considered as continuation of the A line and no new paragraph is started.
- By default, all paragraphs receive the SAPscript standard paragraph format '*'. To specify a different paragraph format, use the parameters PARAGRAPH_AV and PARAGRAPH_F.

Control statements

The system does not convert the following control statements, since they have no equivalents in R/3:

.nlf print the next two lines on top of each other

.psz set the number of lines per page

.lpi define line distance

.prn output control statement for printer

.off set left margin for lines with V attribute

.nam determine name of spool file

.set define value for number and chapter variable

The other R/2 control statements are converted to SAPscript statements as follows:

.ifi *&symbol*	/: I'F *&symbol(I)&* EQ '
.ifn *&symbol*	I/: 'F *&symbol&* NE '
.abs <*n*>	/:PROTECT contents of the next <*n*> lines from R/2 text /: ENDPROTECT
.def *&symbol*=<*value*>	/: DEFINE *&symbol&* = '<*value*>
.pct=<*value*>	/: DEFINE &pct& = '<*value*>'
.top <*text no*>	/: TOP /: INCLUDE <*text no*> /: ENDTOP
.bot <*text no*>	/: BOTTOM /: INCLUDE <*text no*> /: ENDBOTTOM

Symbols

On one hand, symbols are converted to the different syntax of SAPscript; on the other hand, the names of R/2 database symbols that consist of table name/segment ID and field name must be converted to the different table and field names of R/3.

As for the syntax, you can use SAPscript for all formatting options (offset, length, prefix text), except converting a symbol value using a table.

To assign the symbol names of R/2 database symbols to the corresponding R/3 names, use the function module RS3L_CONVERT_FIELDNAME. For more information, see the documentation of this function module.

The system passes the parameters SOURCE_SYSTEM, SOURCE_RELEASE, SOURCE_OBJECT, DESTINATION_TABLE, and IGNORE_ALIASNAMES without further interpretation to this function module.

The R/2 symbols for chapter variables (p0..p3) are not transferred to SAPscript.

Function call:

```
CALL FUNCTION  'CONVERT_TEXT_R2'
     EXPORTING  HEADER_R2          = ?...
                HEADER             = ?...
```

```
           DIRECTION          = 'IMPORT'
           PARAGRAPH_AV       = '* '
           PARAGRAPH_F        = '* '
           INCLUDE_ID         = 'ST  '
           INCLUDE_ZEROS      = 'X'
           INCLUDE_PREFIX     = SPACE
           COMMENT_LINES      = 'X'
           SOURCE_SYSTEM      = 'SAP'
           SOURCE_RELEASE     = SPACE
           SOURCE_OBJECT      = SPACE
           DESTINATION_TABLE  = SPACE
           IGNORE_ALIASNAMES  = SPACE
           ERROR_COMMENTS     = SPACE
IMPORTING  NEWHEADER_R2       =
           NEWHEADER          =
           NEWSTYLE           =
           ERRORLEVEL         =
TABLES     LINES_R2           = ?...
                LINES           = ?...
```

Export parameters:

HEADER_R2	You must enter the text header of the R/2 text. Structure: TEXTH
HEADER	You must enter the text header of the R/3 text. Structure: THEAD
DIRECTION	Specify the conversion direction. At present, the SAP system supports only conversions from R/2 texts to the SAPscript format, that is, the value IMPORT. Default value: 'IMPORT'
PARAGRAPH_AV	Enter the R/3 paragraph format you want to use at the beginning of an R/2 line block with the attributes A or V. If you leave the field empty or if it contains blanks, the system uses the standard paragraph format '*'. Reference field: TLINE-TDFORMAT Default value: '*'

CONVERT_TEXT_R2

PARAGRAPH_F	Enter the R/3 paragraph format you want to use for each R/2 line with the line attribute F. If you leave the field empty or if it contains blanks, the system uses the standard paragraph format '*'. Reference field: TLINE-TDFORMAT Default value: '*'
INCLUDE_ID	If the R/2 texts include standard texts, the system converts them into a SAPscript INCLUDE statement. Use this parameter to specify the ID of the R/3 text. Reference field: THEAD-TDID Defaultwert 'ST'
INCLUDE_ZEROS	When converting texts from the SAP R/2 system, the system transfers the numbers of the included standard texts with a length of eight digits, thus including leading zeroes. Use the parameter to influence the conversion of the leading zeroes in the standard text number. Possible values: 'X' include leading zeroes ' ' delete leading zeroes Default value: 'X' The R/2 text includes the standard text 33. The respective text line looks like this: S 00000033 In SAPscript, this text line is represented by the INCLUDE statement: /: INCLUDE '00000033' OBJECT 'TEXT' ID 'ST' If the parameter INCLUDE_ZEROS contains SPACE, the generated INCLUDE statement looks like this: /: INCLUDE '33' OBJECT 'TEXT' ID 'ST'
INCLUDE_PREFIX	If the R/2 text modules you want to convert include standard texts, you can enter a prefix here, which the system will place before the text number in the INCLUDE statement: Default value: SPACE INCLUDE_PREFIX = 'R2_' R/2 line: S 00001234 R/3 line: /: INCLUDE 'R2_00001234'....

COMMENT_LINES	The parameter determines how to treat R/2 command lines that SAPscript no longer supports. Possible values: 'X' include lines as comment lines ' ' leave out lines Default value: 'X'
SOURCE_SYSTEM	Enter the name of the system from which the text module originates. Usually, it is 'SAP'. Default value: 'SAP' The function module CONVERT_TEXT_R2 directly passes this parameter to the parameter SOURCE_SYSTEM of the internally called function module RS3L_CONVERT_FIELDNAME, which converts the field name. For more information, see the parameter documentation of that function module.
SOURCE_RELEASE	Enter the release of the R/2 text module. The system needs the entry to convert the database symbols of the R/2 text. Default value: SPACE The function module CONVERT_TEXT_R2 directly passes this parameter to the parameter with the same name of the internally called function module RS3L_CONVERT_FIELDNAME, which converts the field name. For more information, see the parameter documentation of that function module.
SOURCE_OBJECT	Enter the migration object within which the field name is converted. Default value: SPACE The function module CONVERT_TEXT_R2 directly passes this parameter to the parameter BMIG_OBJECT of the internally called function module RS3L_CONVERT_FIELDNAME, which converts the field name. For more information, see the parameter documentation of that function module.
DESTINATION_TABLE	Enter the R/3 table to which the converted text belongs. Default value: SPACE The function module CONVERT_TEXT_R2 directly passes this parameter to the parameter R3_TABLE of the internally called function module RS3L_CONVERT_FIELDNAME, which converts the field name. For more information, see the parameter documentation of that function module.

IGNORE_ALIASNAMES	Use the parameter to control the alias name check during the conversion. Possible values: 'X' ignore check ' ' execute check Default value: SPACE The function module CONVERT_TEXT_R2 directly passes this parameter to the parameter with the same name of the internally called function module RS3L_CONVERT_FIELDNAME, which converts the field name. For more information, see the parameter documentation of that function module.
ERROR_COMMENTS	Use the parameter to define whether to include certain errors, which occur during the conversion, as comments into the SAPscript text. The messages usually refer to an error that was detected when converting the subsequent text line. Possible values: 'X' include error message as comment ' ' do not include error messages For each message line, a prefix specifies where the error message originates. For more information, see the respective documentation. CTR2 error message from text conversion routine -> function module CONVERT_TEXT_R2 RS3L error message from field name conversion -> function module RS3L_CONVERT_FIELDNAME If for an error no error text exists, the system displays after the prefix the character chain '?..'. Default value: SPACE

Import parameters:

NEWHEADER_R2	Changed header of the R/2 text. Since at present only the conversion R/2 to R/3 is supported, the system places the text header specified in the parameter HEADER_R2 without changes into this parameter. Structure: TEXTH

NEWHEADER	The parameter returns the text header with the fields changed according to the executed actions. Structure: THEAD
NEWSTYLE	Not filled at present. Reference field: THEAD-TDSTYLE

CONVERT_TEXT_R2

ERRORLEVEL	When converting R/2 texts to R/3 format, several problems may occur. The parameter returns the highest level of the encountered problems. ERRORLEVEL = 0: No problems detected. ERRORLEVEL = 1: • Smaller problems detected, which concern formatting. The actual text is converted without changes. • Character for default separator deleted. • Words split by line feeds concatenated (only for lines with attributes A or V). • The internal call of function module RS3L_CONVERT_FIELDNAME returned error level 1 when converting the field name. ERRORLEVEL = 2: • The system did convert text elements, but cannot ensure that the R/3 system can interpret these texts. • The converted text contained symbols, for which the system cannot ensure that they can be replaced by the respective values in the R/3 system: – symbols that refer to table fields *) – symbols that refer to segment fields *) – symbols converted using the R/2 table 164V *) These errors can occur only if the function module RS3L_CONVERT_FIELDNAME does not exist. • The internal call of function module RS3L_CONVERT_FIELDNAME returned error level 2 when converting a field name. ERRORLEVEL = 3: • The R/2 text contained functions that SAPscript no longer or not yet supports. • R/2 commands:.nlf,.psz,.lpi,.prn,.off,.nam • Conversion of symbol values using a table &symbol(Txxxx) • Chapter counters: &p0... &p3 • The internal call of function module RS3L_CONVERT_FIELDNAME returned error level 3 when converting a field name. ERRORLEVEL = 4: • Conversion error • Statement line truncated: The line length of the R/3 text module is shorter than a converted command. The system truncated the remainder of the line. SAPscript cannot or not correctly interpret the statement. • Invalid reference to standard text: In the R/2 text, a line with line format S occurs whose line contents is no valid text number. • The internal call of function module RS3L_CONVERT_FIELDNAME returned error level 4 or higher when converting a field name. • The internal call of function module RS3L_CONVERT_FIELDNAME

Table parameters:

LINES_R2	The table contains the text lines of the text in R/2 format. Structure: TEXTL
LINES	The table contains the text lines of the text in the R/3 SAPscript format. Structure: TLINE

CONVERT_OTF_MEMORY

If you use the function module OPEN_FORM with the parameter DEVICE=OTF_MEM, the system outputs the SAPscript text into a buffer instead of creating a spool request. The system then formats the text internally for the pseudo device SCREEN and stores the result in the buffer.

To read the buffer, use function module CONVERT_OTF_MEMORY, which returns the contents in the table parameter LINES. The parameter FORMAT determines the format of the text in this table. The maximum line width used in the table in the field LINES-TDLINE is controlled by the parameter MAX_LINEWIDTH.

The only value supported at present for the FORMAT parameter is 'ASCII'.

With format ASCII, the table LINES contains in the field LINES-TDLINE the text to be output (without any control characters), formatted according to the specified maximum line width. The field LINES-TDFORMAT contains the flag for a SAPscript long line with line feed (/=), if you want to indicate the beginning of the line. Lines that contain only the long line flag (=) in this field, are continuation lines of the previous line. In the ASCII format, page breaks are not indicated.

You can use the text returned with format ASCII as SAPscript text by interpreting the field LINES-TDFORMAT as SAPscript format column. Or, you can use the text as RAW text by ignoring the field LINES-TDFORMAT and interpreting each line of the table LINES (field LINES-TDLINE) as one text line.

Function call:

```
CALL FUNCTION  'CONVERT_OTF_MEMORY'
     EXPORTING  FORMAT                = 'ASCII'

                MAX_LINEWIDTH         = 132

     IMPORTING  BIN_FILESIZE          =

     TABLES     LINES                 = ?...

     EXCEPTIONS MEMORY_EMPTY          =

                ERR_MAX_LINEWIDTH     =

                ERR_FORMAT            =

                ERR_CONV_NOT_POSSIBLE =
```

Export parameters:

FORMAT	Define the format into which you want to convert the SAPscript OTF output. At present, you can use only the ASCII format. Default value: 'ASCII'

MAX_LINEWIDTH	Enter the maximum line width a converted text line may have in table LINES. The maximum line width must be between 2 and 132. Default value: 132

Import parameters:

BIN_FILESIZE	For binary format: number of bytes in LINES

Table parameters:

LINES	The table returns the converted text lines. Structure: TLINE

Exceptions:

MEMORY_EMPTY	The memory was empty. Possible reasons: • During formatting, an error occurred and the memory was not filled. • When reading the memory, an error occurred.
ERR_MAX_LINEWIDTH	The line width specified in parameter MAX_LINEWIDTH is not between 2 and 132.
ERR_FORMAT	The parameter FORMAT contains a target format into which SAPscript cannot convert the OTF output.
ERR_CONV_NOT_POSSIBLE	During conversion of the text from SAPscript OTF format into the specified target format, an error occurred. The conversion process is canceled.