ABAP Syntax

Criado por Guest, última alteração por Srihari Hebbar em set 27, 2007

A

ADD for Single Fields

Adds two single fields.

Syntax

ADD <n> TO <m>.

The contents of <n> are added to the contents of <m>. The result is placed in <m>. Equivalent of
<m> = <m> + <n>.

ADD for Sequences of Fields

Adds sequences of fields.

Syntax

ADD <n1> THEN <n2> UNTIL <nz> GIVING <m>.

ADD <n1> THEN <n2> UNTIL <nz> ACCORDING TO <sel> GIVING <m>.

ADD <n1> THEN <n2> UNTIL <nz> TO <m>.

ADD <n1> FROM <m1> TO <mz> GIVING <m>.

If <n1>, <n2>,..., <nz> is a sequence of fields with a uniform gap between each, the same type, and the same length, the fields are added together and the result placed in <m>. The variants allow you to restrict the fields to a partial sequence, to include <m> in the sum, or to perform the operation for a sequence of consecutive fields.

ADD-CORRESPONDING

Adds components of structures.

Syntax

ADD-CORRESPONDING <struc1> TO <struc2>.

Adds together all of the components of structures <struc1> and <struc2> that have identical names, and places the results in the corresponding components of <struc2>.

APPEND

Appends one or more lines to the end of an index table.

Syntax

APPEND <line>|LINES OF <jtab> TO <itab>.

Appends one line <line> or several lines of an internal table <jtab> to the index table <itab>.

ASSIGN

Assigns a field to a field symbol.

Syntax

ASSIGN <f> TO <FS>.

Assigns the data object <f> to the field symbol <FS>, after which, <FS> points to the data object. The pointed brackets are part of the syntax of the field symbol.

AT for Event Blocks

Event keywords for defining event blocks for screen events.

Syntax

AT SELECTION-SCREEN...

AT LINE-SELECTION.

AT USER-COMMAND.

AT PFn.

User actions on selection screens or lists trigger events in the ABAP runtime environment. The event keywords define event blocks, which are called when the corresponding event occurs.

AT for Control Levels

Control level change when you process extracts and internal tables in a loop.

Syntax

AT NEW <f>.

AT END OF <f>.

AT FIRST.

AT LAST.

AT <fg>.

These statements are used in control level processing with extract datasets or internal tables. Each introduces a statement block that you must conclude with the ENDAT statement. The statements between AT and ENDAT are executed whenever the corresponding control level change occurs.

AUTHORITY-CHECK

Checks the authorization of a user.

Syntax

```
AUTHORITY-CHECK OBJECT <object> ID <name1> FIELD <f1>
                ID <name2> FIELD <f2>
               ...
                ID <name10> FIELD <f10>.
```

The statement checks whether the user has all of the authorizations defined in the authorization object <object>. <name1>,..., <name10> are the authorization fields in the object, and <f1>,... ,<f10> are data objects in the program. The value of each data object is checked against the corresponding authorization field.

B

BACK

Relative positioning for output in a list.

Syntax

```
BACK.
```

Positions the list output either in the first column of the first line after the page header on the current page, or in the first column of the first line of a line block if you have previously used the RESERVE statement.

BREAK-POINT

Starts the ABAP Debugger.

Syntax

```
BREAK-POINT.
```

Interrupts program execution and starts the Debugger. This allows you to test your programs by halting them at any point.

C

CALL CUSTOMER-FUNCTION

Calls a customer function module.

Syntax

```
CALL CUSTOMER-FUNCTION <func>...
```

Similar to CALL FUNCTION. The function module that it calls must be programmed and activated by a customer using the modification concept.

CALL FUNCTION

Calls a function module.

Syntax

```
CALL FUNCTION <func> [EXPORTING ... f_i = a_i... ]
                     [IMPORTING ... f_i = a_i... ]
                     [CHANGING  ... f_i = a_i... ]
                     [TABLES    ... f_i = a_i... ]
                     [EXCEPTIONS... e_i = r_i... ]
                   [DESTINATION <dest>]
                   [IN UPDATE TASK]
                   [STARTING NEW TASK]
                   [IN BACKGOUND TASK].
```

Starts a function module, either in the same system or in an external system, depending on the type of call that you use. You can also use update function modules in transactions, and call function modules asynchronously. The remaining additions are used for the parameter interface of the function module, in which you can specify actual parameters and determine how to handle exceptions.

CALL DIALOG

Calls a dialog module.

Syntax

```
CALL DIALOG <dialog> [AND SKIP FIRST SCREEN]
                     [EXPORTING... f_i = a_i... ]
                     [IMPORTING... f_i = a_i... ]
                     [USING itab].
```

Calls the dialog module <dial>. A dialog module is an ABAP program containing a chain of screens. It does not have to be called using a transaction code, and runs in the same SAP LUW as the program that called it. The additions allow you to skip the first screen in the chain and pass actual parameters to and from the parameter interface of the dialog module.

CALL SCREEN

Calls a sequence of screens.

Syntax

```
CALL SCREEN <scr>
     [STARTING AT <X1> <Y1>]
     [ENDING AT <X2> <Y2>].
```

Calls the sequence of screens beginning with screen number <scr>. All of the screens in the chain belong to the same ABAP program. The chain ends when a screen has the next screen number 0. The additions allow you to call a single screen as a modal dialog box.

CALL SELECTION-SCREEN

Calls a selection screen.

Syntax

CALL SELECTION-SCREEN <scr>
[STARTING AT <x_1> <y_1>]
[ENDING AT <x_2> <y_2>].

Calls a user-defined selection screen in a program. Selection screens are processed in the AT SELECTION-SCREEN events. The additions allow you to call a selection screen as a modal dialog box.

CALL TRANSACTION

Calls a transaction.

Syntax

CALL TRANSACTION <tcod>
[AND SKIP FIRST SCREEN]
[USING <itab>].

Calls the transaction <tcod> while retaining the data in the calling program. At the end of the transaction, control returns to the point from which the transaction was called. The additions allow you to skip the first screen of the transaction or pass an internal table for batch input to it.

CASE

Conditional branching.

Syntax

CASE <f>.

Opens a CASE control structure that must conclude with the ENDCASE statement. This allows you to branch to various statement blocks (introduced with the WHEN statement), depending on the contents of the data object <f>.

CATCH

Catches runtime errors.

Syntax

CATCH SYSTEM-EXCEPTIONS <except$_1$> = <rc $_1$>... <except $_n$> = <rc $_n$>.

Introduces a CATCH area, which concludes with an ENDCATCH statement. If a catchable runtime error **<except$_i$>** occurs within this block, the current block terminates immediately, and the program jumps directly to the corresponding ENDCATCH statement, filling SY-SUBRC with <rc $_i$>.

CHECK

Conditional termination of a loop pass or a processing block.

Syntax

CHECK <logexp>.

If the logical expression <logexp> is true, the program continues at the next statement. If, however, <logexp> is false, the current loop pass terminates and the next begins. If the program is not currently processing a loop, the current processing block terminates. There are special forms of the CHECK statement for use with selection tables and in GET event blocks.

CLEAR

Resets a variable to its initial value.

Syntax

CLEAR <f>.

Resets the variable <f>, which may be of any data type, to the initial value defined for that type.

CLOSE DATASET

Closes a file.

Syntax

CLOSE DATASET <dsn>.

Closes a file <dsn> on the application server previously opened with the OPEN DATASET statement.

CLOSE CURSOR

Closes a database cursor.

Syntax

CLOSE CURSOR <c>.

Closes a cursor opened using the OPEN CURSOR statement.

COLLECT

Inserts lines into an internal table in summarized form.

Syntax

COLLECT <line> INTO <itab>.

The statement first checks whether the internal table contains an entry with the same key. If not, it acts like INSERT. If there is already a table entry with the same key, COLLECT does not insert a new line. Instead, it adds the values from the numeric fields of the work area <line> to the values in the corresponding fields of the existing table entry.

COMMIT

Concludes an SAP LUW.

Syntax

COMMIT WORK [AND WAIT].

All database updates are written firmly to the database, and all locks are released. Triggers the database update. The AND WAIT addition allows you to pause the program until the update is complete. If you omit it, the database is updated asynchronously.

COMMUNICATION

Allows communication between programs.

Syntax

COMMUNICATION INIT DESTINATION <dest> ID <id> [Additions].

COMMUNICATION ALLOCATE ID <id> [Additions].

COMMUNICATION ACCEPT ID <id> [Additions].

COMMUNICATION SEND ID <id> BUFFER <f> [Additions].

COMMUNICATION RECEIVE ID <id> [Additions].

COMMUNICATION DEALLOCATE ID <id> [Additions].

These statements allow you to initialize, start, and accept program-to-program communication, send and receive data between partner programs, and then terminate the connection.

COMPUTE

Performs numeric operations.

Syntax

COMPUTE <n> = <expression>.

The result of the mathematical expression in <expression> is assigned to the result field <n>. The COMPUTE keyword is optional.

CONCATENATE

Combines a series of strings into a single string.

Syntax

CONCATENATE <c1>... <cn> INTO <c> [SEPARATED BY <s>].

The strings <c1> to <cn> are concatenated, and the result placed in <c>. The SEPARATED BY addition allows you to specify a string <s> to be placed between the strings.

CONDENSE

Removes spaces from a string.

Syntax

CONDENSE <c> [NO-GAPS].

Removes all leading spaces, and replaces other series of blanks with a single space in the character field <c>. If you use the NO-GAPS addition, all of the spaces are removed.

CONSTANTS

Declares constant data objects.

Syntax

CONSTANTS <c>... VALUE [<val> | IS INITIAL]...

The syntax is similar to DATA, except that the VALUE addition is required, and that internal tables and deep structures cannot be declared as constants. The starting value that you assign in the VALUE addition cannot be changed during the program.

CONTINUE

Ends a loop pass.

Syntax

CONTINUE.

Only possible within loops. This statement terminates the current loop pass and starts the next.

CONTEXTS

Declares a context.

Syntax

CONTEXTS <c>.

Generates an implicit data type CONTEXT_<c>, which you can use to create context instances.

CONTROLS

Defines a control.

Syntax

CONTROLS <ctrl> TYPE <ctrl_type>.

Defines an ABAP runtime object <ctrl>. This displays data in a particular format on a screen, depending on the type <ctrl_type>. Currently, <ctrl_type> may be a table control or tabstrip control.

CONVERT for Dates

Converts a data into an inverted date form.

Syntax

CONVERT DATE <d1> INTO INVERTED-DATE <d2>.

CONVERT INVERTED-DATE <d1> INTO DATE <d2>.

If <d1> and <d2> are date fields in the internal form YYYYMMDD, the nines complement of <d1> is placed in field <d2> and vice versa. In inverted date format, the most recent date has the smaller numerical value.

CONVERT for Timestamps

Converts a timestamp into the correct date and time for the current time zone.

Syntax

CONVERT TIME STAMP <tst> TIME ZONE <tz> INTO DATE <d> TIME <t>.

CONVERT DATE <d> TIME <t> INTO TIME STAMP <tst> TIME ZONE <tz>.

As long as <tst> has type P(8) or P(11) with 7 decimal placed, and <tz> has type C(6), the time stamp <tst> will be converted to the correct date <d> and time <t> for the time zone <tz>.

CONVERT for Text

Converts a text into a format that can be sorted alphabetically.

Syntax

CONVERT TEXT <text> INTO SORTABLE CODE <x>.

<text> must have type C and <x> must have type X. The string is then converted so that the relative order of the characters allows them to be sorted alphabetically in the current text environment.

D

DATA with Reference to Declared Data Types

Declares variables with a previously-declared data type.

Syntax

DATA <f>... [TYPE <type>|LIKE <obj>]... [VALUE <val>].

Declares a variable <f> with the fully-defined data type <type> or the same data type as another data object <obj>. The data type <type> can be D, F, I, T, a type defined locally in the program using the TYPES statement, or a type from the ABAP Dictionary. The data object <obj> is a data object or line of an internal table that has already been defined. The VALUE addition allows you to specify a starting value.

DATA with Reference to Generic Data Types

Declares variables by completing the description of a generic type.

Syntax

DATA <f>[(<length>)] TYPE <type> [DECIMALS <d>]... [VALUE <val>].

DATA <f> TYPE <itab>.

The data type <type> can be C, N, P, or X. In the <length> option, you specify the length of the field. If you do not specify the length, the default value for the data type is used. If <type> is P, you can use the DECIMALS addition to specify a number of decimal places <d>. If you do not specify a number of decimal places, it is set to none. If you do not specify a type, the system uses the default type C.

Syntax

DATA <f> TYPE <itab>.

The data type <itab> is a standard internal table with generic key. The default key is automatically used in the DATA statement.

DATA, Creating an Associated Data Type

Declares variables with data types that only exist as an attribute of the variable.

Syntax

DATA <f> TYPE REF TO <class>|<interface>.

Declares the variable <f> as a reference variable for the class <class> or the interface <interface>.

Syntax

DATA: BEGIN OF <structure>,
...
<f_i>...,
...
END OF <structure>.

Combines the variables <f_i> to form the structure <structure>. You can address the individual components of a structure by placing a hyphen between the structure name and the component name: <structure>-<f $_i$>.

Syntax

DATA <f> TYPE|LIKE <tabkind> OF <linetype> WITH <key>.

Declares the variable <f> as an internal table with the table type <tabkind>, line type <linekind>, and key <key>.

DATA for Shared Data Areas

Declares shared data areas in a program.

Syntax

DATA: BEGIN OF COMMON PART <c>,
<f $_i$>. **..**
END OF COMMON PART.

The variables <f_i> are assigned to a data area <c>, which can be defined in more than one program. These data areas use the same memory addresses for all programs that are loaded into the same internal session.

DEFINE

Defines a macro.

Syntax

DEFINE <macro>.

Introduces the definition of the macro <macro>. Each macro must consist of complete ABAP statement and be concluded with the END-OF-DEFINITION statement.

DELETE for Files

Deletes files on the application server.

Syntax

DELETE DATASET <dsn>.

Deletes the file <dsn> from the file system of the application server.

DELETE for Database Table Entries

Deletes entries from database tables.

Syntax

DELETE FROM <dbtab> WHERE <cond>.

Deletes all of the lines from the database table <dbtab> that satisfy the WHERE condition.

Syntax

DELETE <dbtab> FROM <wa>.

DELETE <dbtab> FROM TABLE <itab>.

Deletes the lines with the same primary key as the work area <wa>, or all of the lines from the database table with the same primary key as one of the lines in the internal table <itab>. The work area <wa> or the lines of the internal table <itab> must be at least as long as the primary key of the database table and have the same alignment.

DELETE for Cluster Database Tables

Deletes data clusters from cluster database tables.

Syntax

DELETE FROM DATABASE <dbtab>(<ar>) ID <key>.

Deletes the entire data cluster from the area <ar> with the name <key> from the cluster database table <dbtab>.

DELETE for the Cross-Transaction Application Buffer

Deletes data clusters from the cross-transaction application buffer.

Syntax

DELETE FROM SHARED BUFFER <dbtab>(<ar>) ID <key>.

Deletes the data cluster for the area <ar> with the name <key> stored in the cross-transaction application buffer for the table <dbtab>.

DELETE for Lines from an Internal Table

Deletes lines from internal tables of any type.

Syntax

DELETE TABLE <itab> FROM <wa>.

DELETE TABLE <itab> WITH TABLE KEY $<k_1>$ = $<f_1>$... $<k_n>$ = $<f_n>$.

Deletion using the table key: All lines with the same key are deleted. The key values are taken either from a compatible work area <wa> or specified explicitly.

Syntax

DELETE <itab> WHERE <cond>.

Deletion using a condition: Deletes all table entries that satisfy the logical expression <cond>. The logical condition may consist of more than one expression. However, the first operand in each expression must be a component of the line structure.

Syntax

DELETE ADJACENT DUPLICATE ENTRIES FROM <itab> [COMPARING...].

Deletes adjacent duplicate entries, either by comparing the key fields or the comparison fields specified explicitly in the COMPARING addition.

DELETE for Lines from Index Tables

Deletes lines from index tables.

Syntax

DELETE <itab> [INDEX <idx>].

If you use the INDEX option, deletes the line with the index <idx> from the table <itab>. If you do not use the INDEX option, the statement can only be used within a LOOP ... ENDLOOP construction. In this case, it deletes the current line.

Syntax

DELETE <itab> [FROM $<n_1>$] [TO $<n_2>$] [WHERE <cond>].

Deletes all rows from <itab> with index between $<n_1>$ and $<n_2>$ an which satisfy the WHERE condition. If you do not use the FROM addition, the system deletes lines starting at the beginning of the table. If you do not use the TO addition, the system deletes lines to the end of the table. The logical expression <cond> can consist of more than one expression. However, the first operand in each expression must be a component of the line structure of the internal table.

DEMAND

Retrieves values from a context instance.

Syntax

DEMAND <val$_1$> = <f $_1$>... <val $_n$> = <f $_n$> FROM CONTEXT <inst>
[MESSAGES INTO <itab>].

Fills the fields <f$_n$> with the values <val $_n$> of the context instance <inst>. The MESSAGES addition allows you to control how messages from the context are handled in the program.

DESCRIBE DISTANCE

Determines the distance between two fields.

Syntax

DESCRIBE DISTANCE BETWEEN <f1> AND <f2> INTO <f3>.

Writes the distance in bytes between fields <f1> and <f2> to <f3>, always including the length of the field that occurs first in memory.

DESCRIBE FIELD

Describes the attributes of a field.

Syntax

DESCRIBE FIELD <f> [LENGTH <l>] [TYPE <t> [COMPONENTS <n>]]
[OUTPUT-LENGTH <o>] [DECIMALS <d>]
[EDIT MASK <m>] [HELP-ID <h>].

The attributes of the data object <f> named in the additions to the statement are placed in the corresponding variables. You can use any number of additions in a single statement.

DESCRIBE LIST

Describes the attributes of a list.

Syntax

DESCRIBE LIST NUMBER OF LINES <lin> [INDEX <idx>].

DESCRIBE LIST NUMBER OF PAGES <n> [INDEX <idx>].

DESCRIBE LIST LINE <lin> PAGE <pag> [INDEX <idx>].

DESCRIBE LIST PAGE <pag> [INDEX <idx>]...

Depending on the variant of the statement that you use, writes the number of lines, number of pages, a line of a list on a given page, or various attributes of a page to variables.

DESCRIBE TABLE

Describes the attributes of an internal table.

Syntax

DESCRIBE TABLE [LINES <l>] [OCCURS<n>] [KIND <k>].

Depending on the additions you use, writes the number of lines occupied, the value specified for the INITIAL SIZE of the table, or the table type into a corresponding variable.

DIVIDE

Divides one field by another.

Syntax

DIVIDE <n> BY <m>.

Divides the content of <n> by <m>, and places the result in <n>. The equivalent of n = n / m.

DIVIDE-CORRESPONDING

Divides matching components of structures.

Syntax

DIVIDE-CORRESPONDING <struc1> BY <struc2>.

Divides all matching components of the structures <struc1> and <struc2> and places the results into the corresponding components of <struc1>.

DO

Introduces a loop.

Syntax

DO [<n> TIMES] [VARYING <f> FROM <f1> NEXT <f2>].

Introduces a statement block that must conclude with ENDDO. If you omit the TIMES addition, the statement block is repeated until a termination statement such as CHECK or EXIT occurs. The TIMES addition restricts the number of loop passes to <n>. The VARYING addition allows you to process a sequence of fields the same distance apart in memory.

E

EDITOR-CALL

Loads an ABAP program or internal table into a text editor.

Syntax

EDITOR-CALL FOR <itab>...

EDITOR-CALL FOR REPORT <prog>...

Loads the internal table <itab> or the program <prog> into a text editor, where you can edit it using standard editor functions.

EXEC SQL

Introduces a Native SQL statement.

Syntax

EXEC SQL [PERFORMING <form>].

Between EXEC SQL and the ENDEXEC statement, you can include a database-specific Native SQL statement. The PERFORMING addition allows you to pass a multiple-line selection line by line to a subroutine.

EXIT

Terminates a loop or processing block.

Syntax

EXIT.

Within a loop: The entire loop is terminated, and processing continues with the first statement following the loop.

Outside a loop: Terminates the current processing block.

In a reporting event: Jumps directly to the output list.

EXIT FROM STEP-LOOP

Ends a step loop.

Syntax

EXIT FROM STEP-LOOP.

Terminates step loop processing. A step loop is a way of displaying a table on a screen.

EXIT FROM SQL

Terminates Native SQL processing.

Syntax

EXIT FROM SQL.

This statement may occur within a subroutine called using the PERFORMING addition in the EXEC SQL statement. The entire subroutine is processed, but no more subsequent lines of the selection are processed.

EXPORT

Exports a data cluster.

Syntax

EXPORT... <f_i> [FROM <g_i>]... | (<itab>)

TO MEMORY
| DATABASE <dbtab>(<ar>) ID(<key>)
| SHARED BUFFER <dbtab>(<ar>) ID(<key>).

The data objects <f_i> or <g_i>, or the data objects in the internal table <itab> are stored as a data cluster in the cross-program ABAP memory of the current internal session, in a cluster database table <dbtab>, or in the cross-transaction application buffer of the table <dbtab>.

EXTRACT

Creates an extract dataset and adds lines to it.

Syntax

EXTRACT <fg>.

The first EXTRACT statement in a program creates an extract dataset and adds the first entry to it. Each subsequent EXTRACT statement adds a new entry. Each extract entry contains the fields of the field group <fg> and, at the beginning, the fields of the field group HEADER as a sort key.

F

FETCH

Uses a cursor to read entries from a database table.

Syntax

FETCH NEXT CURSOR <c> INTO <target>.

If the cursor <c> is linked with a selection in a database table, FETCH writes the next line of the selection into the flat target area <target>.

FIELD-GROUPS

Declares a field group for an extract dataset.

Syntax

FIELD-GROUPS <fg>.

Declares the field group <fg>. Field groups define the line structure of an extract dataset. You can define a special field group called HEADER as the sort key. When you fill the extract dataset, the HEADER field group precedes each entry.

FIELD-SYMBOLS

Declares a field symbol.

Syntax

FIELD-SYMBOLS <FS> [<type>|STRUCTURE <s> DEFAULT <wa>].

Field symbols are placeholders or symbolic names for fields. The pointed brackets in the name of a field symbol are part of its syntax. The <type> addition allows you to specify a type. The STRUCTURE addition imposes a structure on the data object assigned to the field symbol.

FORM

Defines a subroutine.

Syntax

FORM <subr> [USING ... [VALUE()<p_i>[])] [TYPE <t>|LIKE <f>]...]
[CHANGING... [VALUE()<p_i>[])] [TYPE <t>|LIKE <f>]...].

Introduces a subroutine <form>. The USING and CHANGING additions define the parameter interface. The subroutine definition is concluded with the ENDFORM statement.

FORMAT

Sets formatting options for list output.

Syntax

FORMAT... <option$_i$> [ON|OFF]...

The formatting options <option$_i$> (color, for example) apply to all subsequent list output until they are disabled with the OFF option.

FREE

Releases memory space.

Syntax

FREE <itab>.

FREE MEMORY ID(<key>).

This statement deletes an internal table, a data cluster in ABAP memory, or an external object in OLE2 Automation, depending on the form of the statement used. It also releases the memory occupied by the object.

FUNCTION

Defines a function module.

Syntax

FUNCTION <func>.

Introduces the function module <func>. This statement does not have to be entered in the ABAP Editor, but is automatically generated by the Function Builder in the ABAP Workbench. The function module definition is concluded with the ENDFUNCTION statement.

FUNCTION-POOL

Introduces a function group.

Syntax

FUNCTION-POOL.

The first statement in a function group. This statement does not have to be entered by hand, but is generated automatically by the Function Builder in the ABAP Workbench. A function group is an ABAP program that contains function modules.

G

GET

Event keyword that defines event blocks for reporting events.

Syntax

GET <node> [FIELDS <f_1> <f_2>...].

Only occurs in executable programs. When the logical database has passed a line of the node <node> to the program, the runtime environment triggers the GET event, and the corresponding event block is executed. The FIELDS addition allows you to specify explicitly the columns of the node that the logical database should retrieve.

GET BIT

Reads an individual bit.

Syntax

GET BIT <n> OF <f> INTO <g>.

Reads the bit at position <n> of the hexadecimal field <f> into the field <b>.

GET CURSOR

Determines the cursor position on a screen or in an interactive list event.

Syntax

GET CURSOR FIELD <f> [OFFSET <off>]
[LINE <lin>]
[VALUE <val>]
[LENGTH <len>].

GET CURSOR LINE <lin> [OFFSET <off>]
[VALUE <val>]
[LENGTH <len>].

At a user action on a list or screen, the statement writes the position, value, and displayed length of a field or line into the corresponding variables.

GET LOCALE LANGUAGE

Finds out the current text environment.

Syntax

GET LOCALE LANGUAGE <lg> COUNTY <c> MODIFIER <m>.

Returns the current language, country ID and any modifier into the corresponding variables.

GET PARAMETER

Finds out the value of a SPA/GPA parameter.

Syntax

GET PARAMETER ID <pid> FIELD <f>.

Places the value of the SPA/GPA parameter <pid> from the user-specific SAP memory into the variable <f>.

GET PF-STATUS

Finds out the current GUI status.

Syntax

GET PF-STATUS <f> [PROGRAM <prog>] [EXCLUDING <itab>].

Returns the name of the current GUI status (the same as SY-PFKEY) into the variable <f>. The PROGRAM addition writes the name of the ABAP program to which the status belongs into the variable <prog>. The EXCLUDING addition returns a list of all currently inactive function codes into the internal table <itab>.

GET RUN TIME FIELD

Measures the runtime in microseconds.

Syntax

GET RUN TIME FIELD <f>.

The first time the statement is executed, the variable <f> is set to zero. In each further call, the runtime since the first call is written to <f>.

GET TIME

Synchronizes the time.

Syntax

GET TIME [FIELD <f>].

Refreshes the system fields SY-UZEIT, SY-DATUM, SY-TIMLO, SY-DATLO, and SY-ZONLO. If you use the FIELD addition, the variable <f> is filled with the current time.

GET TIME STAMP FIELD

Returns a time stamp.

Syntax

GET TIME STAMP FIELD <f>.

Returns the short or long form of the current date and time, depending on whether the variable <f> has the type P(8) or P(11). The long form returns the time correct to seven decimal places.

H

HIDE

Stores information about list lines.

Syntax

HIDE <f>.

During list creation, this statement stores the contents of the field <f> and the current line number in the internal HIDE area. When the cursor is positioned on a line in an interactive list event, the stored value is returned to the field <f>.

I

IF

Conditional branching.

Introduces a new branch.

Syntax

IF <logexp>.

Opens an IF control structure that must be concluded with ENDIF. The system evaluates the logical expression <logexp>, and processes different statement blocks depending on the result.

IMPORT

Imports a data cluster.

Syntax

```
IMPORT... <fi> [TO <g i>]... | (<itab>)
    FROM  MEMORY
      | DATABASE <dbtab>(<ar>) ID(<key>)
      | SHARED BUFFER <dbtab>(<ar>) ID(<key>).
```

The data objects <fi> or the objects listed in the table <itab> are written from data clusters in the cross-program ABAP memory of the current internal session, a cluster database table <dbtab>, or the cross-transaction application buffer of the table <dbtab> into the variables <f i> or <g i>.

IMPORT DIRECTORY

Creates the directory of a data cluster from a cluster database.

Syntax

```
IMPORT DIRECTORY INTO <itab>
    FROM DATABASE <dbtab>(<ar>)
    Id <key>.
```

The statement creates a directory of the data objects in a data cluster of the cluster database <dbtab> and writes it to the internal table <itab>.

In the third variant, the table <itab> contains a directory of the objects stored using EXPORT TO DATABASE.

INCLUDE

Inserts an include program in another program.

Syntax

INCLUDE <incl>.

This has the same effect as inserting the source code of the include program <incl> at the same position in the program as the INCLUDE statement. Includes are not loaded dynamically at runtime, but are automatically expanded when the program is loaded. An include must have the program type I.

INCLUDE STRUCTURE

Includes a structure within another.

Syntax

INCLUDE STRUCTURE <s>|TYPE <t>.

Adopts the structure of an ABAP Dictionary structure <s> or a structured data type <t> as part of a new structure declared using DATA BEGIN OF ...

INITIALIZATION

Event keyword that defines an event block for a reporting event.

Syntax

INITIALIZATION.

Only occurs in executable programs. The ABAP runtime environment triggers the INITIALIZATION event before the selection screen is processed, at which point the corresponding event block is processed.

INSERT for Database Tables

Inserts lines into a database table.

Syntax

INSERT <dbtab> FROM <wa>.

INSERT <dbtab> FROM TABLE <itab> [ACCEPTING DUPLICATE KEYS].

Inserts one line from the work area <wa> or several lines from the internal table <itab> into the database table <dbtab>. The addition ACCEPTING DUPLICATE KEYS prevents a runtime error from occurring if two entries have the same primary key. Instead, it merely discards the duplicate.

INSERT for Field Groups

Defines the structure of field groups for extract datasets.

Syntax

INSERT $<f_1>$... $<f_n>$ INTO <fg>.

Includes the fields $<f_i>$ in the field group <fg>, thus defining a line structure for an extract dataset.

INSERT for any Internal Table

Inserts lines in an internal table of any type.

Syntax

INSERT <line>|LINES OF <jtab> [FROM $<n_1>$] [TO $<n_2>$]
INTO TABLE <itab>.

Inserts a line <line> or a set of lines from the internal table <jtab> into the internal table <itab>. If <jtab> is an index table, you can use the FROM and TO additions to restrict the lines inserted.

INSERT for Index Tables

Inserts lines in index tables.

Syntax

INSERT <line>|LINES OF <jtab> [FROM $<n_1>$] [TO $<n_2>$]
INTO <itab> [INDEX <idx>].

Inserts a line <line> or a set of lines from an internal table <jtab> into the internal table <itab> before the line with the index <idx>. If <jtab> is an index table, you can restrict the lines to be inserted using the FROM and TO additions. If you omit the INDEX addition, you can only use the statement within a LOOP construction. In this case, the new line is inserted before the current line.

INSERT for Programs

Inserts ABAP programs into the program library.

Syntax

INSERT REPORT <prog> FROM <itab>.

The lines of the internal table <itab> are added to the program library as the program <prog>.

L

LEAVE for Screens

Leaves a screen.

Syntax

LEAVE SCREEN.

Terminates the current screen and calls the next screen. The next screen can either be defined statically in the screen attributes or set dynamically using the SET SCREEN statement.

Syntax

LEAVE TO SCREEN <scr>.

Terminates the current screen and calls the dynamically-defined next screen <scr>.

LEAVE for Lists During Screen Processing

Switches between screen and list processing.

Syntax

LEAVE TO LIST-PROCESSING [AND RETURN TO SCREEN <scr>].

This statement allows you to create and display a list while processing a series of screens. The addition allows you to specify the next screen (to which you return after the list has been displayed). If you do not use the addition, screen processing resumes with the PBO of the current screen.

Syntax

LEAVE LIST-PROCESSING.

Allows you to switch back explicitly from list processing to screen processing.

LEAVE for Programs

Terminates an ABAP program.

Syntax

LEAVE [PROGRAM].

Terminates the current program and returns to the point from which it was called.

Syntax

LEAVE TO TRANSACTION <tcod> [AND SKIP FIRST SCREEN].

Terminates the current program and starts a new transaction <tcod>. The addition allows you to skip the initial screen of the transaction.

LOCAL

Protects global data against changes.

Syntax

LOCAL <f>.

Only occurs in subroutines. When the subroutine starts, the value of <f> is stored temporarily, and restored to the variable <f> at the end of the subroutine.

LOOP Through Extracts

Starts a loop through an extract dataset.

Syntax

LOOP.

Loops through an extract dataset. The loop is concluded with ENDLOOP. When the LOOP statement is executed, the system finishes creating the extract dataset, and loops through all of its entries. One entry is read in each loop pass. The values of the extracted data are placed in the output fields of the field group within the loop.

LOOP Through Internal Tables

Starts a loop through an internal table.

Syntax

LOOP AT <itab> INTO <wa> WHERE <logexp>.

LOOP AT <itab> ASSIGNING <FS> WHERE <logexp>.

LOOP AT <itab> TRANSPORTING NO FIELDS WHERE <logexp>.

Loops through an internal table. The loop is concluded with ENDLOOP. If the logical expression <logexp> is true, the current line contents are either placed in the work area <wa>, assigned to the field symbol <FS>, or not assigned at all. The first operand in each part of <logexp> must be a component of the internal table. The pointed brackets in the field symbol name are part of its syntax.

With index tables, you can use the additions FROM <n> and TO <n> to restrict the lines that are read by specifying an index range.

LOOP Through Screen Fields

Starts a loop through the special table SCREEN.

Syntax

LOOP AT SCREEN...

Similar to a loop through an internal table. The system table SCREEN contains the names and attributes of all of the fields on the current screen.

M

MESSAGE

Outputs a message.

Syntax

MESSAGE <xnnn> [WITH <f1>... <f4>] [RAISING <except>].

MESSAGE ID <mid> TYPE <x> NUMBER <nnn>.

MESSAGE <xnnn>(<mid>).

Outputs the message <nnn> of message class <mid> as message type <x>. The message type determines how the message is displayed, and how the program reacts. The WITH addition allows you to fill placeholders in the message text. The RAISING addition in function modules and methods allows you to terminate the procedure and trigger the exception <exception>.

MODIFY for Database Tables

Inserts or changes lines in database tables.

Syntax

MODIFY <dbtab> FROM <wa>.

MODIFY <dbtab> FROM TABLE <itab>.

Works like INSERT for database tables if there is not yet a line in the table with the same primary key. Works like UPDATE if a line already exists with the same primary key.

MODIFY for All Internal Tables

Changes the contents of lines in any type of internal table.

Syntax

MODIFY TABLE <itab> FROM <wa> [TRANSPORTING $<f_1>$ $<f_2>$...].

Copies the work area <wa> into the line of the internal table with the same table key as <wa>. You can use the TRANSPORTING addition to specify the exact components that you want to change.

MODIFY <itab> FROM <wa> TRANSPORTING $<f_1>$ $<f_2>$... WHERE <logexp>.

Copies the work area <wa> into the lines of the internal table for which the logical expression is true. The first operand in each comparison of the logical expression must be a component of the line structure.

MODIFY for Index Tables

Changes the contents of lines in index tables.

Syntax

MODIFY <itab> FROM <wa> [INDEX <idx>] [TRANSPORTING $<f_1>$ $<f_2>$...].

Copies the work area <wa> into the line of the internal table with index <idx>. If you omit the INDEX addition, you can only use the statement within a LOOP. This changes the current line.

MODIFY for Lists

Changes a line of a list.

Syntax

```
MODIFY LINE <n> [INDEX <idx>] [OF CURRENT PAGE|OF PAGE <p>]
    |CURRENT LINE
    LINE FORMAT <option1> <option2>...
    FIELD VALUE <f1> [FROM <g1>] <f2> [FROM <g2>]...
    FIELD FORMAT <f1> <options1> <f2> <options2>
```

Changes either line <n> on the current or specified list (or page), or the last line to be chosen. The exact nature of the change is specified in the additions.

MODIFY SCREEN

Changes the table SCREEN.

Syntax

MODIFY SCREEN...

Like changing an internal table. This statement allows you to change the attributes of fields on the current screen.

MODULE

Introduces a dialog module.

Syntax

MODULE <mod> OUTPUT |[INPUT].

Introduces the dialog module <mod>. The OUTPUT and INPUT additions designate the module as a PBO or PAI module respectively. Each module must conclude with the ENDMODULE statement.

MOVE

Assigns values.

Syntax

MOVE <f1> TO <f2>.

Assigns the contents of the data object <f1> to the variable <f2>, with automatic type conversion if necessary. Equivalent to <f2> = <f1>.

MOVE-CORRESPONDING

Assigns values between identically-named components of structures.

Syntax

MOVE-CORRESPONDING <struc1> TO <struc2>.

The contents of the components of the structure <struc1> are assigned to the identically-named components of the structure <struc2>.

MULTIPLY

Multiplies two individual fields.

Syntax

MULTIPLY <n> BY <m>.

The contents of <n> are multiplied by the contents of <m>, and the result is placed in <m>. Equivalent is M = m * n.

MULTIPLY-CORRESPONDING

Multiplies components of structures.

Syntax

MULTIPLY-CORRESPONDING <struc1> BY <struc2>.

Multiplies all of the identically-named components of <struc1> and <struc2> and places the results in the components in <struc1>.

N

NEW-LINE

Inserts a line break in a list.

Syntax

NEW-LINE [NO-SCROLLING|SCROLLING].

Positions the list output on a new line. The NO-SCROLLING addition locks the line against horizontal scrolling. To lift the lock, use the SCROLLING addition.

NEW-PAGE

Inserts a page break in a list.

Syntax

NEW-PAGE [NO-TITLE|WITH-TITLE]
[NO-HEADING|WITH-HEADING]
[LINE-COUNT]
[LINE-SIZE]
[PRINT ON|OFF].

Generates a new page and positions the list output after the page header. The additions control how the page header is displayed, the length and width of the page, and the print output.

NODES

Declares an interface work area.

Syntax

NODES <node>.

Declares a variable with the same data type and the same name as a data type from the ABAP Dictionary. Structures in main programs and subroutines declared with nodes use a common data area. This statement is used in conjunction with logical databases.

O

ON CHANGE

Introduces a branch.

Syntax

ON CHANGE OF <f> [OR <f1> OR <f2>...].

Opens an ON control structure, concluded with ENDON. The statement block is executed whenever the contents of the field <f> or one of the other fields <fi> has changed since the statement was last executed.

OPEN CURSOR

Opens a database cursor.

Syntax

```
OPEN CURSOR [WITH HOLD] <c> FOR SELECT     <result>
                        FROM     <source>
                        [WHERE    <condition>]
                        [GROUP BY <fields>]
                        [HAVING   <cond>]
                        [ORDER BY <fields>].
```

Opens a cursor <c> with type CURSOR for a SELECT statement. All of the clauses of the SELECT statement apart from the INTO clause can be used. The INTO clause is set in the FETCH statement. If you use the WITH HOLD addition, the cursor is not closed when a database commit occurs.

OPEN DATASET

Opens a file.

Syntax

OPEN DATASET <dsn> [FOR INPUT|OUTPUT|APPENDING]
[IN BINARY|TEXT MODE]
[AT POSITION <pos>]
[MESSAGE <mess>]
[FILTER <filt>].

Opens a file <dsn> on the application server. The additions determine whether the file is for reading or writing, whether the contents are to be interpreted in binary or character form, the position in the file, the location to which an operating system can be written, and allow you to execute an operating system command.

OVERLAY

Overlays a character string with another.

Syntax

OVERLAY <c1> WITH <c2> [ONLY <str>].

All of the characters in field <c1> that occur in <str> are overlaid with the contents of <c2>. <c2> remains unchanged. If you omit the ONLY <str> addition, all positions in <c1> containing spaces are overwritten.

P

PACK

Converts type C variables into type P.

Syntax

PACK <f> TO <g>.

Packs the string <f> and places it in the field <g>. This can be reversed with the UNPACK statement.

PARAMETERS

Declares parameters for a selection screen.

Syntax

PARAMETERS <p>[(<length>)] [TYPE <type>|LIKE <obj>] [DECIMALS <d>]
[DEFAULT <f>]
[MEMORY ID <pid>]
[LOWER CASE]
[OBLIGATORY]
[VALUE CHECK]
[AS CHECKBOX]
[RADIOBUTTON GROUP <radi>]
[NO-DISPLAY]
[MODIF ID <key>].

Declares a variable <p>, as in the DATA statement. However, the PARAMETERS statement also creates an input field for <p> on the relevant selection screen. You can use the additions to define default values, accept lowercase input, define the field as required, check values, define a checkbox or radio button, prevent the field from being displayed on the selection screen, or modify the field.

PERFORM

Calls a subroutine.

Syntax

PERFORM <subr>
| <subr>(<prog>) [IF FOUND]
|(<fsubr>)[IN PROGRAM (<fprog>)][IF FOUND]
[USING ... <p_i>...]
[CHANGING... <p_i>...]
[ON COMMIT].

Calls an internal or external subroutine <subr> or the subroutine whose name occurs in the <fsubr> field. The external program is <prog> or the name contained in <fprog>. The IF FOUND addition prevents a runtime error from occurring if the subroutine does not exist. You must use the USING and CHANGING additions to supply values to the interface parameters of the subroutine. The ON COMMIT addition delays the execution of the subroutine until the next COMMIT WORK statement.

POSITION

Absolute positioning of the output on a list.

Syntax

POSITION <col>.

Positions the list output in column <col>.

PRINT-CONTROL for Print Format

Specifies the print format.

Syntax

PRINT-CONTROL <formats> [LINE <lin>] [POSITION <col>].

Sets the print format starting either at the current list position or at line <lin> and column <col>.

PRINT-CONTROL for Index Lines

Creates index lines in the spool file.

Syntax

PRINT-CONTROL INDEX-LINE <f>.

Writes the contents of the field <f> into an index line at the end of the current print line. The index line is not printed. In optical archiving, the spool system separates the lists into a data file and a description file containing the index lines.

PROGRAM

Introduces a program.

Syntax

PROGRAM <prog>...

The first statement in some ABAP programs. Equivalent of REPORT.

PROVIDE

Loops through internal tables at given intervals.

Syntax

PROVIDE <f1>... <fn> FROM <itab1>
<g1>... <gm> FROM <itab2>
... FROM <itabn>
... BETWEEN <f> AND <g>.

The contents of the specified fields of the internal tables <itab1> ... <itabn> are placed in their header lines. Then, the processing block between PROVIDE and ENDPROVIDE is executed for each interval.

PUT

Triggers a GET event.

Syntax

PUT <node>.

Only occurs in logical databases. Branches the program flow according to the structure of the logical database.

R

RAISE for Exceptions

Triggers an exception.

Syntax

RAISE <except>.

Only occurs in function modules and methods. Terminates the procedure and triggers the exception <except>.

RANGES

Declares a RANGES table.

Syntax

RANGES <rangetab> FOR <f>.

Declares a RANGES table for the field <f>. RANGES tables have the same data type as a selection table, but they do not have input fields on a selection screen.

READ for Files

Reads a file.

Syntax

READ DATASET <dsn> INTO <f> [LENGTH <len>].

Reads the contents of the file <dsn> on the application server to the variable <f>. The number of bytes transferred can be written to <len>.

READ for any Internal Table

Reads a line from any internal table.

Syntax

```
READ TABLE <itab>  FROM <wa>
             |WITH TABLE KEY <k1> = <f1>... <kn> = <fn>
             |WITH KEY = <f>
             |WITH KEY <k1> = <f1>... <kn> = <fn>
   INTO <wa> [COMPARING <f1> <f2>... |ALL FIELDS]
          [TRANSPORTING <f1> <f2>... |ALL FIELDS|NO FIELDS]
  |ASSIGNING <FS>.
```

This statement reads either the line of the internal table with the same key as specified in the work area <wa>, the line with the key specified in the TABLE KEY addition, the line that corresponds fully to <f>, or the one corresponding to the freely-defined key in the KEY addition. The contents of the line are either written to the work area <wa>, or the line is assigned to the field symbol <FS>. If you assign the line to a work area, you can compare field contents and specify the fields that you want to transport.

READ for Index Tables

Reads a line of an index table.

Syntax

READ TABLE <itab> INDEX <idx> INTO <wa>... | ASSIGNING <FS>.

Reads the line with the index <idx>. The result is available as described above.

READ for Lists

Reads the contents of a line from a list.

Syntax

```
READ LINE  <n> [INDEX <idx>] [OF CURRENT PAGE|OF PAGE <p>]
      |CURRENT LINE
      [FIELD VALUE <f1> [INTO <g1>]... <fn> [INTO <gn>]].
```

Reads either the line <n> on the current or specified list or page, or the last line to have been selected by the user. The addition specifies the fields that you want to read, and the target fields into which they should be placed. The entire line is always placed in the system field SY-LISEL, and the HIDE area is filled for the line.

READ for Programs

Reads ABAP programs from the program library.

Syntax

READ REPORT <prog> INTO <itab>.

Copies the lines of the program <prog> into the internal table <itab>.

RECEIVE

Receives results from an asynchronous function module call.

Syntax

RECEIVE RESULTS FROM FUNCTION <func> [KEEPING TASK]
[IMPORTING ... $f_i = a_i$...]
[TABLES ... $f_i = a_i$...]
[EXCEPTIONS... $e_i = r_i$...]

Occurs in special subroutines to receive IMPORTING and TABLES parameters from function modules called using the STARTING NEW TASK addition.

REFRESH

Initializes an internal table.

Syntax

REFRESH <itab>.

Resets the internal table <itab> to its initial value, that is, deletes all of its lines.

REFRESH CONTROL

Initializes a control.

Syntax

REFRESH CONTROL <ctrl> FROM SCREEN <scr>.

The control <ctrl> defined in the CONTROLS statement is reset with the initial values specified for screen <scr>.

REJECT

Terminates a GET processing block.

Syntax

REJECT [<dbtab>].

Stops processing the current line of the node of the logical database. If you specify <dbtab>, the logical database reads the next line of the node <dbtab>, otherwise the next line of the current node.

REPLACE

Replaces strings in fields with another string.

Syntax

REPLACE <str1> WITH <str2> INTO <c> [LENGTH <l>].

This statement searches the first occurrence of the first <l> characters of the search pattern <str1> in field <c> and replaces them with the string <str2>.

REPORT

Introduces a program.

Syntax

REPORT <rep> [MESSAGE-ID <mid>]
[NO STANDARD PAGE HEADING]
[LINE-SIZE <col>]
[LINE-COUNT <n>(<m>)]
[DEFINING DATABASE <ldb>].

This is the first statement within certain ABAP programs. <rep> can be any name you choose. The addition MESSAGE-ID specifies a message class to be used in the program. The DEFINING DATABASE addition defines the program as the database program of the logical database <ldb>. The other options are formatting specifications for the default list of the program.

RESERVE

Conditional page break in a list.

Syntax

RESERVE <n> LINES.

Executes a page break on the current page if less than <n> lines are free between the current line and the page footer.

ROLLBACK

Undoes the changes in a SAP LUW.

Syntax

ROLLBACK WORK.

Undoes all changes within a database LUW to the beginning of the LUW. The registered update modules are not executed, and the record entry is deleted from table VBLOG.

S

SCROLL

Scrolls in a list.

Syntax

SCROLL LIST FORWARD|BACKWARD [INDEX <idx>].

SCROLL LIST TO FIRST PAGE|LAST PAGE|PAGE <pag>
[INDEX <idx>] [LINE <lin>].

SCROLL LIST LEFT|RIGHT [BY <n> PLACES] [INDEX <idx>].

SCROLL LIST TO COLUMN <col> [INDEX <idx>].

Positions the current list or the list level <idx> in accordance with the additions specified. You can scroll by window, page, columns, or to the left- or right-hand edge of the list.

SEARCH

Searches for a string.

Syntax

SEARCH <f>|<itab> FOR <g> [ABBREVIATED]
[STARTING AT <n1>]
[ENDING AT <n2>]
[AND MARK].

Searches the field <f> or the table <itab> for the string in field <g>. The result is placed in the system field SY-FDPOS. The additions allow you to hide intermediate characters, search from and to a particular position, and convert the found string into uppercase.

SELECT

Reads data from the database.

Syntax

```
SELECT <result>
  INTO <target>
  FROM <source>
  [WHERE <condition>]
  [GROUP BY <fields>]
  [HAVING <cond>]
  [ORDER BY <fields>].
```

The SELECT statement consists of a series of clauses, each of which fulfils a certain task:

SELECT clause

Defines the structure of the selection.

Syntax

```
SELECT [SINGLE]|[DISTINCT]
      * | <s_i> [AS <a_i>]... <agg>( [DISTINCT] <s_j>) [AS <a_j>]...
```

The selection can be a single line SINGLE or a series of lines. You can eliminate duplicate lines using the DISTINCT addition. To select the entire line, use *, otherwise, you can specify individual columns <s_i>. For individual columns, you can use aggregate functions <agg>, and assign alternative column names <a_i>.

INTO clause

Defines the target area into which the selection from the SELECT clause is to be placed.

Syntax

```
...  INTO [CORRESPONDING FIELDS OF] <wa>
  | INTO|APPENDING [CORRESPONDING FIELDS OF] TABLE <itab>
                                  [PACKAGE SIZE <n>]
  | INTO (<f_1>, <f_2>,...)
```

The target area can be a flat work area <wa>, an internal table <itab>, or a list of fields <f_i>. If you use the CORRESPONDING FIELDS addition, data is only selected if there is an identically-named field in the target area. If you use APPENDING instead of INTO, the data is appended to an internal table instead of overwriting the existing contents. PACKAGE SIZE allows you to overwrite or extend the internal table in a series of packages. The data type of the target area must be appropriate for the selection in the SELECT clause.

FROM clause

Specifies the database tables from which the data in the selection in the SELECT clause is to be read.

Syntax

```
... FROM [<tab> [INNER]|LEFT [OUTER] JOIN] <dbtab> [AS <alias>]
                        [ON <cond>]
       [CLIENT SPECIFIED]
       [BYPASSING BUFFER]
       [UP TO <n> ROWS]
```

You can read a single table <dbtab> or more than one table, using inner and outer joins to link tables with conditions <cond>, where <tab> is a single table or itself a join condition. You can specify individual database tables either statically or dynamically, and you can replace their names with an alternative <alias>. You can bypass automatic client handling with the CLIENT SPECIFIED addition, and SAP buffering with BYPASSING BUFFER. You can also restrict the number of lines read from the table using the UP TO <n> ROWS addition.

WHERE clause

Restricts the number of lines selected.

Syntax

```
... [FOR ALL ENTRIES IN <itab>] WHERE <cond>
```

The condition <cond> may contain one or more comparisons, tests for belonging to intervals, value list checks, subqueries, selection table queries or null value checks, all linked with AND, OR, and NOT. If you use the FOR ALL ENTRIES addition, the condition <cond> is checked for each line of the internal table <itab> as long as <cond> contains a field of the internal table as an operand. For each line of the internal table, the lines from the database table meeting the condition are selected. The result set is the union of the individual selections resulting from each line.

GROUP BY clause

Groups lines in the selection

Syntax

```
... GROUP BY <s_1> <s_2>
```

Groups lines with the same contents in the specified columns. Uses aggregate functions for all other columns in each group. All columns listed in the SELECT clause that do not appear in the GROUP BY addition must be specified in aggregate expressions.

HAVING clause

Restricts the number of line groups.

Syntax

```
... HAVING <cond>
```

Like the WHERE clause, but can only be used in conjunction with a GROUP BY clause. Applies conditions to aggregate expressions to reduce the number of groups selected.

ORDER BY clause

Sorts the lines in the selection.

Syntax

... ORDER BY PRIMARY KEY |... <s_i> [ASCENDING|DESCENDING]...

Sorts the selection in ascending or descending order according to the primary key or the contents of the fields listed.

SELECT-OPTIONS

Declares selection criteria for a selection screen.

Syntax

```
SELECT-OPTIONS <sel> FOR <f>
        [DEFAULT <g> [to <h>] [OPTION <op>] SIGN <s>]
        [MEMORY ID <pid>]
        [LOWER CASE]
        [OBLIGATORY]
        [NO-DISPLAY]
        [MODIF ID <key>]
        [NO-EXTENSION]
        [NO INTERVALS]
        [NO DATABASE SELECTION].
```

Declares a selection table <sel> for the field <f>, and also places input fields on the corresponding selection screen. The additions allow you to set a default value, accept input in lowercase, define a required field, suppress or modify the display on the selection screen, restrict the selection table to a line or a selection to a single field, or prevent input from being passed to a logical database.

SELECTION-SCREEN for Selection Screen Formatting

Formats a selection screen.

Syntax

```
SELECTION-SCREEN SKIP [<n>].

SELECTION-SCREEN ULINE [[/]<pos(len)>] [MODIF ID <key>].

SELECTION-SCREEN COMMENT [/]<pos(len)> <comm> [FOR FIELD <f>]
                              [MODIF ID <key>].

SELECTION-SCREEN BEGIN OF LINE.
 ...
SELECTION-SCREEN END OF LINE.

SELECTION-SCREEN BEGIN OF BLOCK <block>
                    [WITH FRAME [TITLE <title>]]
                    [NO INTERVALS].
 ...
SELECTION-SCREEN END OF BLOCK <block>.

SELECTION-SCREEN FUNCTION KEY <i>.

SELECTION SCREEN PUSHBUTTON [/]<pos(len)> <push>
                  USER-COMMAND <ucom> [MODIF ID <key>].
```

Allows you to insert blank lines, lines and comments, group input fields together in lines and blocks, and create pushbuttons.

SELECTION-SCREEN for Defining Selection Screens

Defines selection screens.

Syntax

```
SELECTION-SCREEN BEGIN OF <numb> [TITLE <tit>] [AS WINDOW].
 ...
SELECTION-SCREEN END OF <numb>.
```

Defines a selection screen with the screen number <numb>. All PARAMETERS, SELECT-OPTIONS, and SELECTION-SCREEN statements within the SELECTION-SCREEN BEGIN OF ... END OF block belong to the selection screen <numb>. The TITLE addition allows you to define a title for the selection screen. The AS WINDOW addition allows you to define the selection screen as a modal dialog box.

SELECTION-SCREEN for Selection Screen Versions

Defines selection screen versions.

Syntax

```
SELECTION-SCREEN BEGIN OF VERSION <dynnr>
 ...
 SELECTION-SCREEN EXCLUDE <f>.
 ...
SELECTION-SCREEN BEGIN OF VERSION <dynnr>.
```

Only in logical databases. The statement hides fields that otherwise appear on the standard selection screen.

SELECTION-SCREEN for Logical Databases

Provides special functions in conjunction with logical databases.

Syntax

```
SELECTION-SCREEN DYNAMIC SELECTIONS | FIELD SELECTION
          FOR NODE|TABLE <node>.
```

Can only be used in logical databases. Declares a node as accepting dynamic selections or field selections.

SET BIT

Sets an individual bit.

Syntax

SET BIT <n> OF <f> [TO <b>].

Sets t he bit at position <n> of hexadecimal field <f> to 1 or the value of the field <b>. <b> must be 0 or 1.

SET BLANK LINES

Allows blank lines in lists.

Syntax

SET BLANK LINES ON|OFF.

Prevents blank lines created in WRITE statements from being suppressed in list output.

SET COUNTRY

Sets the output format

Syntax

SET COUNTRY <c>.

Sets the output formats for numeric and date fields for the country with the ID <c>.

SET CURSOR

Sets the cursor on the screen.

Syntax

SET CURSOR FIELD <f> [OFFSET <off>]
[LINE <lin>].

SET CURSOR LINE <lin> [OFFSET <off>].

SET CURSOR <col> <line>.

Sets the cursor either to a particular position in a field, line, or column of a line.

SET EXTENDED CHECK

Affects the extended program check.

Syntax

SET EXTENDED CHECK ON|OFF.

Switches the extended program check (SLIN) on or off, suppressing the corresponding messages.

SET HOLD DATA

Sets a screen attribute.

Syntax

SET HOLD DATA ON|OFF.

Sets the screen attribute "Hold data" from the program.

SET LANGUAGE

Sets the display language.

Syntax

SET LANGUAGE <lg>.

All text symbols are refreshed with the contents of the text pool in language <lg>.

SET LEFT SCROLL BOUNDARY

Sets the left-hand boundary for horizontal scrolling.

Syntax

SET LEFT SCROLL-BOUNDARY [COLUMN <col>].

Sets the current output position or the position <col> as the left-hand edge of the scrollable area on the current list page.

SET LOCALE LANGUAGE

Sets the text environment.

Syntax

SET LOCALE LANGUAGE <lg> [COUNTRY <c>] [MODIFIER <m>].

Sets the text environment for alphabetical sorting according to the language <lg>, country <c>, and any further modifier <m>.

SET MARGIN

Sets the margin of a print page.

SET MARGIN <x> [<y>].

Sends the current list page to the spool system with a margin of <x> columns from the left-hand edge and <y> rows from the top edge of the page.

SET PARAMETER

Sets a SPA/GPA parameter.

Syntax

SET PARAMETER ID <pid> FIELD <f>.

Copies the value of the field <f> into the SPA/GPA parameter <pid> in the user-specific SAP memory.

SET PF-STATUS

Sets the GUI status.

Syntax

```
SET PF-STATUS <stat> [EXCLUDING <f>|<itab>]
            [IMMEDIATELY] [OF PROGRAM <prog>].
```

Sets the GUI status <stat> for the subsequent screens. The EXCLUDING addition allows you to deactivate functions dynamically. The IMMEDIATELY addition sets the GUI status of the list currently displayed. The OF PROGRAM addition allows you to use a GUI status from another program.

SET RUN TIME ANALYZER

Controls the runtime analysis.

Syntax

```
SET RUN TIME ANALYZER ON|OFF.
```

The runtime analysis only measures the runtime of the statements in the block between SET RUN TIME ANALYZER ON and OFF.

SET RUN TIME CLOCK

Sets the clock accuracy for runtime analysis.

Syntax

```
SET RUN TIME CLOCK RESOLUTION HIGH|LOW.
```

Sets the accuracy of the runtime to low accuracy with long measurement interval or high accuracy with shorter measurement interval.

SET SCREEN

Sets the next screen.

Syntax

```
SET SCREEN <scr>.
```

Temporarily overwrites the statically-defined next screen with <scr>. <scr> is processed after the current screen.

SET TITLEBAR

Sets the screen title.

Syntax

```
SET TITLEBAR <t> [OF PROGRAM <prog>] [WITH <g1 >... <g9>].
```

Sets the title <t> for the subsequent screens. The OF PROGRAM addition allows you to use a title from another program. The WITH addition fills any placeholders in the title.

SET UPDATE TASK LOCAL

Switches on local update.

Syntax

```
SET UPDATE TASK LOCAL.
```

Updates are processed in the current work process.

SET USER-COMMAND

Triggers a list event.

Syntax

```
SET USER-COMMAND <fc>.
```

Triggers a list event with the function code <fc> and calls the corresponding event block.

SHIFT

Shifts strings.

Syntax

```
SHIFT <c> [BY <n> PLACES] [LEFT|RIGHT|CIRCULAR].
```

Shifts the field <c> by one or <n> places. The additions allow you to specify the direction, and how the empty spaces are dealt with.

SKIP for Blank Lines

Creates blank lines on the output list.

Syntax

SKIP [<n>].

Creates <n> blank lines after the current line in a list. If you omit <n>, inserts one line.

SKIP for Positioning

Absolute positioning for output on a list.

Syntax

SKIP TO LINE <lin>.

Positions the list output in line <lin>.

SORT for Extracts

Sorts an extract dataset.

Syntax

SORT [ASCENDING|DESCENDING] [AS TEXT] [STABLE]
... BY <f_i> [ASCENDING|DESCENDING] [AS TEXT]...

Ends creation of the extract dataset in the program and sorts it. If you omit the BY addition, the extract is sorted by the key specified in the HEADER field group. The BY addition allows you to specify a different sort key. The other additions specify whether you want to sort in ascending or descending order, and whether strings should be sorted alphabetically.

SORT for Internal Tables

Sorts internal tables.

Syntax

SORT <itab> [ASCENDING|DESCENDING] [AS TEXT] [STABLE]
... BY <f_i> [ASCENDING|DESCENDING] [AS TEXT]...

Sorts the internal table <itab>. If you omit the BY addition, the table is sorted by its key. The BY addition allows you to specify a different sort key. The remaining additions allow you to specify whether you want to sort in ascending or descending order, and whether strings should be sorted alphabetically.

SPLIT

Splits a character string.

Syntax

SPLIT <c> AT <del> INTO <c1>... <cn> INTO TABLE <itab>.

Searches the field <c> for the character <del> and places the partial fields before and after <del> into the target fields <c1> ... <cn>, or into a new line of the internal table <itab>.

START-OF-SELECTION

Event keyword that defines an event block for a reporting event.

Syntax

START-OF-SELECTION.

After the selection screen has been processed, the runtime environment triggers the START-OF-SELECTION event and processes the corresponding event block.

STATICS

Defines static variables.

Syntax

STATICS <f>...

Like DATA. Retains the value of a local variable beyond the runtime of the procedure in which it occurs.

STOP

Exits a reporting event.

Syntax

STOP.

Can only occur in event blocks for reporting events. Leaves the block and jumps to END-OF-SELECTION.

SUBMIT

Calls an executable program.

Syntax

SUBMIT <rep> [AND RETURN] [VIA SELECTION-SCREEN]
[USING SELECTION-SET <var>]
[WITH <sel> <criterion>]
[WITH FREE SELECTIONS <freesel>]
[WITH SELECTION-TABLE <rspar>]
[LINE-SIZE <width>]
[LINE-COUNT <length>].

Calls the program <rep>. If you omit the AND RETURN addition, the current program is terminated, otherwise, the data from the current program is retained, and processing returns to the calling program when <rep> has finished running. The other additions control the selection screen and set attributes of the default list in the called program.

SUBTRACT for Single Fields

Subtracts two single fields.

Syntax

SUBTRACT <n> FROM <m>.

The contents of <n> are subtracted from the contents of <m> and the result placed in <m>. Equivalent of m = m - n.

SUBTRACT-CORRESPONDING

Subtracts components of structures.

Syntax

SUBTRACT-CORRESPONDING <struc1> FROM <struc2>.

Subtracts the contents of the components of <struc1> from identically-named components in <struc2> and places the results in the components of <struc2>.

SUM

Calculates sums of groups.

Syntax

SUM.

Can only be used in loops through internal tables. Calculates the sums of the numeric fields in all lines of the current control level and writes the results to the corresponding fields in the work area.

SUPPLY

Fills context instances with values.

Syntax

SUPPLY <key$_1$> = <f $_1$>... <key $_n$> = <f $_n$> TO CONTEXT <inst>.

Fills the key fields <key$_n$> of the context instance <inst> with the values <f $_n$>.

SUPPRESS DIALOG

Prevents the current screen from being displayed.

Syntax

SUPPRESS DIALOG.

Can only occur in a PBO dialog module. The screen is not displayed, but its flow logic is still processed.

T

TABLES

Declares an interface work area.

Syntax

TABLES <dbtab>.

Declares a structure with the same data type and name as a database table, a view, or a structure from the ABAP Dictionary. Structures declared using TABLES in main programs and subroutines use a common data area.

TOP-OF-PAGE

Event keyword for defining an event block for a list event.

Syntax

TOP-OF-PAGE [DURING LINE-SELECTION].

Whenever a new page begins while a standard list is being created, the runtime environment triggers the TOP-OF-PAGE event and the corresponding event block is executed. The addition DURING LINE-SELECTION has the same function, but for detail lists.

TRANSFER

Writes data to a file.

Syntax

TRANSFER <f> TO [LENGTH <len>].

Writes the field <f> to the file <dsn> on the application server. The LENGTH addition specifies the length <len> of the data you want to transfer.

TRANSLATE

Converts characters in strings.

Syntax

TRANSLATE <c> TO UPPER|LOWER CASE
|USING <r>.

The characters of the string <c> are converted into upper- or lowercase, or according to a substitution rule specified in <r>.

TYPE-POOL

Introduces a type group.

Syntax

TYPE-POOL <tpool>.

The first statement in a type group. You do not have to enter this statement in the ABAP Editor - instead, it is automatically inserted in the type group by the ABAP Dictionary. A type group is an ABAP program containing type definitions and constant declarations that can then be used in several different programs.

TYPE-POOLS

Declares the types and constants of a type group to a program.

Syntax

TYPE-POOLS <tpool>.

After this statement, you can use all of the data types and constants defined in the type group <tpool> in your program.

TYPES for Simple Field Types

Defines a simple field type.

Syntax

TYPES <t>[(<length>)] [TYPE <type>|LIKE <obj>] [DECIMALS <dec>].

Defines the internal data type <t> in the program with length <len>, reference to the ABAP Dictionary type <type> or a data object <obj>, and, where appropriate, with <dec> decimal places.

Syntax

TYPES <t> TYPE REF TO <class>|<interface>.

Defines the internal data type <t> in the program with reference to the class <class> or the interface <interface>.

TYPES for Aggregate Types

Defines aggregated types.

Syntax

TYPES: BEGIN OF <structure>,
...
<t_i>...,
...
END OF <structure>.

Combines the data types <t_i> to form the structure <structure>. You can address the individual components of a structure in a program using a hyphen between the structure name and the component name as follows: <structure>-<t_i>.

Syntax

TYPES <t> TYPE|LIKE <tabkind> OF <linetype> [WITH <key>].

Defines the local data type <t> in the program as an internal table with the access type <tabkind>, the line type <linetype>, and the key <key>.

U

ULINE

Places a horizontal line on the output list.

Syntax

ULINE [AT [/][<pos>][(<len>)]].

Without additions, generates a new line on the current list and fills it with a horizontal line. The additions allow you to insert a line break and specify the starting position and length of the line.

UNPACK

Converts variables from type P to type C.

Syntax

UNPACK <f> TO <g>.

Unpacks the packed field <f> and places it in the string <g> with leading zeros. The opposite of PACK.

UPDATE

Modifies lines in database tables.

Syntax

UPDATE <dbtab> SET <s_i> = <f>
|<s_i> = <s_i> + <f>
|<s_i> = <s_i> - <f> [WHERE <cond>].

Sets the value in <s_i> to <f>, increases it by <f>, or decreases it by <f> for all selected lines. The WHERE addition determines the lines that are updated. If you omit the WHERE addition, all lines are updated.

Syntax

UPDATE <dbtab> FROM <wa>.

UPDATE <dbtab> FROM TABLE <itab>.

Overwrites the line with the same primary key as <wa> with the contents of <wa>, or all lines with the same primary key as a line in the internal table <itab> with the corresponding line of itab. The work area <wa> or the lines of the table <itab> must have at least the same length and the same alignment as the line structure of the database table.

W

WHEN

Introduces a statement block in a CASE control structure.

Syntax

WHEN <f_1> [OR <f_2> OR...] | OTHERS.

The statement block after a WHEN statement is executed if the contents of the field <f> in the CASE statement are the same as those of one of the fields <f_i>. Processing then resumes after the ENDCASE statement. The WHEN OTHERS statement block is executed if the contents of <f> do not correspond to any of the fields <f_i>.

WHILE

Introduces a loop.

Syntax

WHILE <logexp> [VARY <f> FROM <f1> NEXT <f2>].

Introduces a statement block that is concluded with ENDWHILE. The statement block between WHILE and ENDWHILE is repeated for as long as the expression <logexp> is true, or until a termination statement such as CHECK or EXIT occurs. The VARY addition allows you to process fields that are a uniform distance apart within memory.

WINDOW

Displays a list as a modal dialog box.

Syntax

WINDOW STARTING AT <x1> <y1> [ENDING AT <x2> <y2>].

Can only be used in list processing. The current detail list is displayed as a modal dialog box. The top left-hand corner of the window is positioned at column <x1> and line <y1>. The bottom right-hand corner is positioned at column <x2> and line <y2> (if specified).

WRITE

Creates list output.

Syntax

```
WRITE [AT [/][<pos>][(<len>)]] <f> [AS CHECKBOX|SYMBOL|ICON|LINE]
                  [QUICKINFO <g>].
                  [<format>]
```

The contents of the field <f> are formatted according to their data type and displayed on the list. The additions before the field allow you to specify a line break, the starting position, and the length of the field. The additions after the field allow you to display checkboxes, symbols, icons, and lines. The <format> addition can contain various other formatting options. The QUICKINFO addition allows you to assign a quickinfo <g> to the field.

WRITE TO

Assigns strings.

Syntax

```
WRITE <f1> TO <f2> [<format>].
```

Converts the contents of the data object <f1> to type C and assigns the resulting string to the variable <f2>. You can use the same formatting options available in the WRITE statement.

Sem r\u00f3tulos

1 Página Filha

list of alv